Week 10: Machine Learning at Scale:

Large scale graph processing Random walks, PageRank,Personalized pagerank

Based on slides by:

James G. Shanahan[1,2]

[1]Church and Duncan Group Inc., [2]iSchool UC Berkeley, CA,

Aptronym - Wikipedia

Housekeeping

HW update

- HW4: Linear regression and regularized LR

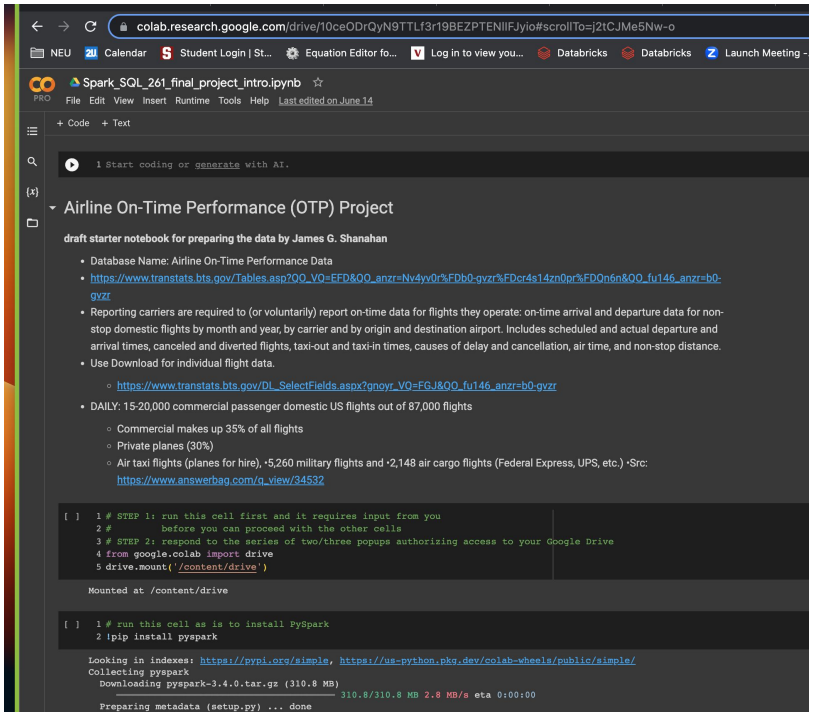
- HW5: due Week 11
  - Pagerank
  - Complete on Google cloud

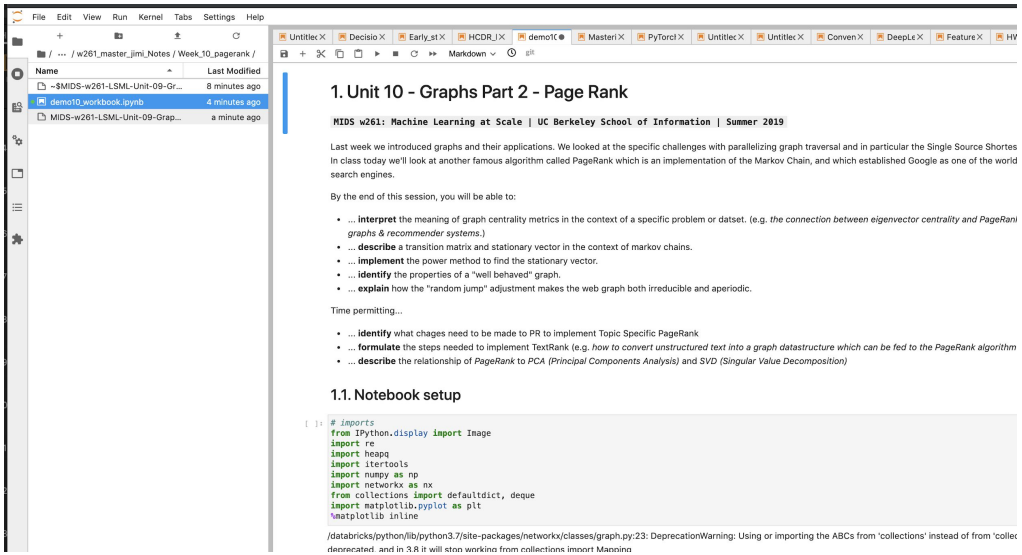
Important Links for Module 10 live session

Slides for live session 10:
https://docs.google.com/presentation/d/1OW35d46rDXxj4gsRxQy4ryw_c4O4Wqw07Fm_bwJTBP0/edit?usp=sharing

Google Colab Notebook for Day 1 of the flights and weather data (plus PageRank in PySpark):
https://colab.research.google.com/drive/10ceODrQyN9TTLf3r19BEZPTENlIFJyio?usp=sharing

Dataproc notebook: see demo_10 on DataProc





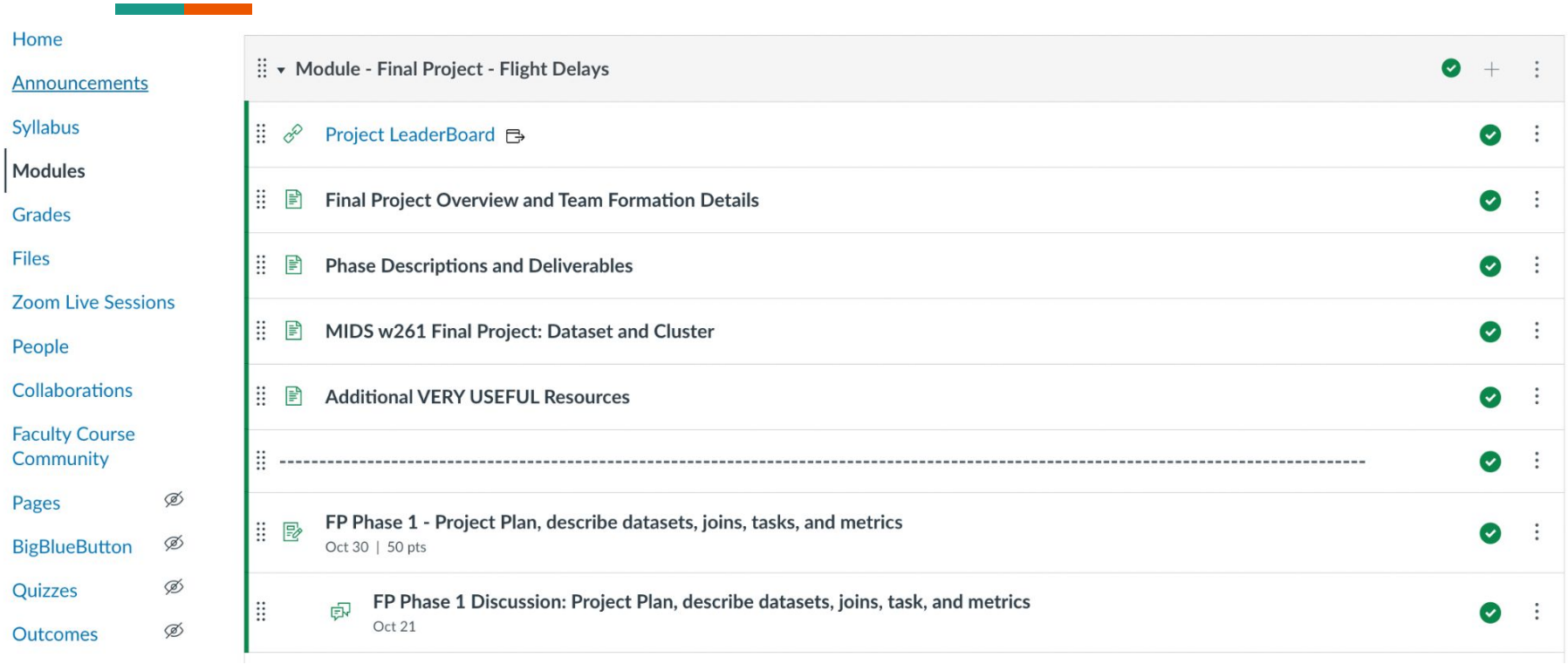
- Once teams are formed, everyone needs to fill in their team details on DC under the **"People"** section. You can join a group by adding your nam to one of the final project groups in DC. Click the **"People"** TAB on the left-hand navigation menu, and select the **"FP_SectionK_GroupN"** tab t view available groups. Each group is limited to a max of four students inside DC.
- Please see here to learn more about joining a group in DC (do not create your own student groups, you must join a pre-setup group created by your Instructors for this project).
- Please use the team following team name prefix, *FP_SectionK_**GroupN**_* as is but feel free to add a team suffix such as *RootFinders* to yield a tea name of *FP_Section1_Group2-RootFinders; when K denotes the section you belong to and N denotes a unique group number with your section.*

For more details on the final project, please see Module - Final Project - Flight Delays.

Schedule at a glance

Phase Description (For details please see detailed phase descriptions and deliverables)	Submission details (or just see *Module - Final Project - Flight Delays*)	Week number	DataBricks cluster details
Phase 0: Finalize Teams (teams of four people)	Due Sunday of Week 9	Week 9	Available from Monday/Tues week 10 (One master and one worker)
Phase I - project plan, describe datasets, joins, tasks, and metrics	Phase 1 Update: Notebook. Due Sunday of Week 10	Week 10	
Phase II - EDA, Baseline Pipeline on all data: Scalability, Efficiency, Distributed/parallel Training, and Scoring Pipeline	Phase 2 Update: Video (2 minutes max) + Notebook. Due Sunday of Week 11	Week 11	AutoScaling enabled from Monday of this week (up to 5)
Phase III Feature engineering + hyperparameter tuning, + **in-class review**	Phase 3 update: Notebook. Due by Sunday of Week 12	Week 12	
Phase IV - Advanced model architectures and loss functions, select an optimal algorithm, fine-tune & Final report write up	Phase 4 update: 2-minute Video + Notebook due by Sunday of Week 13	Week 13	AutoScaling enabled from Monday of this week (up to 10 depending on budget) NOTE: Clusters will be scaled back on Sunday of this week.
Phase V - **In-class Presentation**	Slides uploaded prior to in-class presentation.	Week 14	

Review Final Project and timeline: 5 Phases



Home
Announcements
Syllabus
Modules
Grades
Files
Zoom Live Sessions
People
Collaborations
Faculty Course Community
Pages
BigBlueButton
Quizzes
Outcomes
Discussions
Assignments
Rubrics
Settings

View All Pages

Published | Edit

MIDS w261 Final Project: Dataset and Cluster

Domestic (US and US territories) Flight Delay Project

MIDS w261: Machine Learning at Scale | UC Berkeley School of Information

DataBricks Spark Cluster

Project Onboarding on Databricks PaaS: Cluster, blob storage, starter notebook (available at end of week 9 on Sunday)

- Starter notebook
  - Final project starter notebook
- Databricks US-WEST 2 Region
- Azure Account (for data storage)
  - Create an account (get $100 credit)
  - Create blob storage in the US West region to avoid excessive data transfer fees
  - Checkpoint intermediate results such as data transformations (e.g., joins) to avoid excessive data transfer fees (avoid rerunning all steps in the pipeline)
  - Desktop app/CLI
  - For more details on how to manage your cloud storage on Azure please see:
    - https://azure.microsoft.com/en-us/features/storage-explorer/#overview
    - For more background on Blob Storage please see this notebook: Blob storage Notebook,
      - Details on setting up an account and blob storage
      - and on checkpointing your data

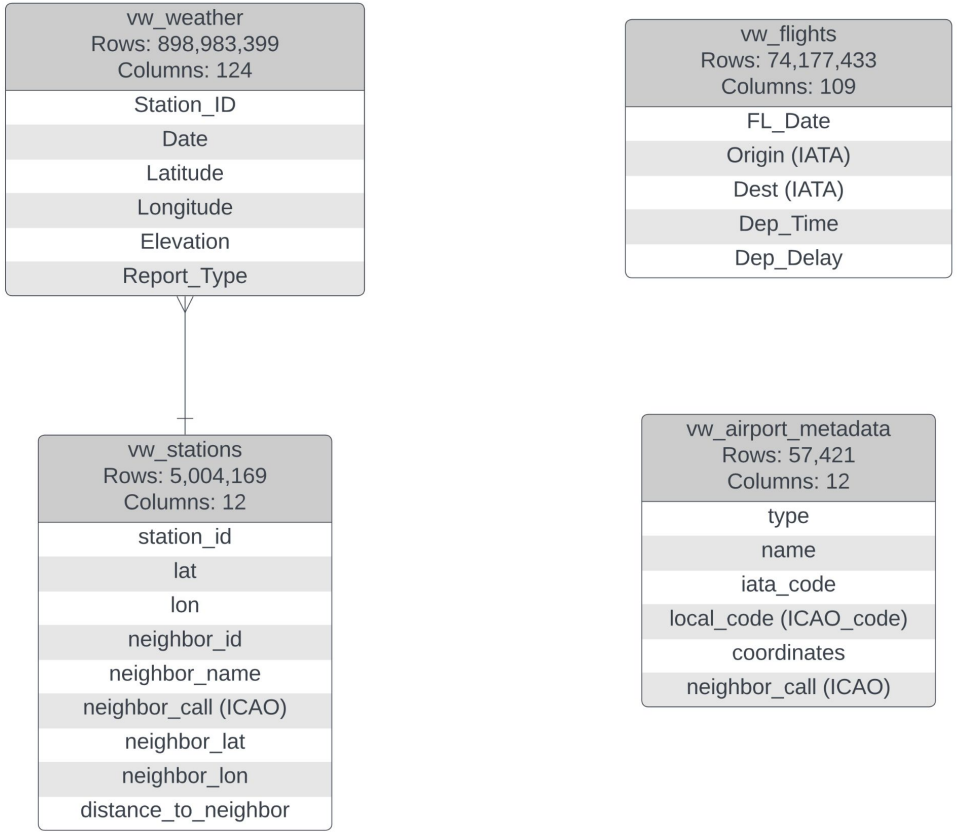
Starter notebook

- **Final project starter notebook**

Flights Data

```
# 7 years has 42,430,592 flights (while weather has 35,532,883 records)
# 3 months of df_flights has 1,420,255 rows, 88 cols
# 1 month of df_flights has. 500,000 flights; one day has 15-20,000 flights
# 12 months of df_flights has 5,832,098 rows, 88 cols
# df_flights for 36M months has 17,111,358 rows, 88 cols
# df_flights for 24M months has 14,618,075 rows, 91 cols
```

ERD of original table



Flights and weather data

- 7 years of data [1/2015 – 12/2021]
  - 84 months (100 approx.)
- df_flights (USA) has 74,177,433 rows, 109 cols (with duplicates)
- df_weather has 898,983,399 rows, 124 cols (10^9 X 10 ^3) = 10^12
- df_stations has 5,004,169 rows, 12 cols
- df_airport_metadata has 57,421 rows, 12 cols

df_weather goes from 1 Billion rows to 35 Million rows

Flights and weather data (back of envelop stats)

- 7 years of data [1/2015 – 12/2021]
  - 84 months (100 approx.)
- df_flights has 74,177,433 rows, 109 cols 1 Million Per Month 10^6 x 10
- df_weather has 898,983,399 rows, 124 cols 10 Million PM.(84 gig of data);
  - 1 TB of data: 10^9 x1000 = 1 TB of data
- df_stations has 5,004,169 rows, 12 cols
- df_airport_metadata has 57,421 rows, 12 cols

```
sql = """
    select min(FL_DATE), max(FL_DATE) from vw_flights;
    """
df_dates = spark.sql(sql)
display(df_dates)
sql = """
    select min(DATE), max(DATE) from vw_weather
    """
df_dates = spark.sql(sql)
display(df_dates)
```

df_dates: pyspark.sql.dataframe.DataFrame = [min(DATE): string, max(DATE): string]

Table +

	min(FL_DATE)	max(FL_DATE)
1	2015-01-01	2021-12-31 00:00:00

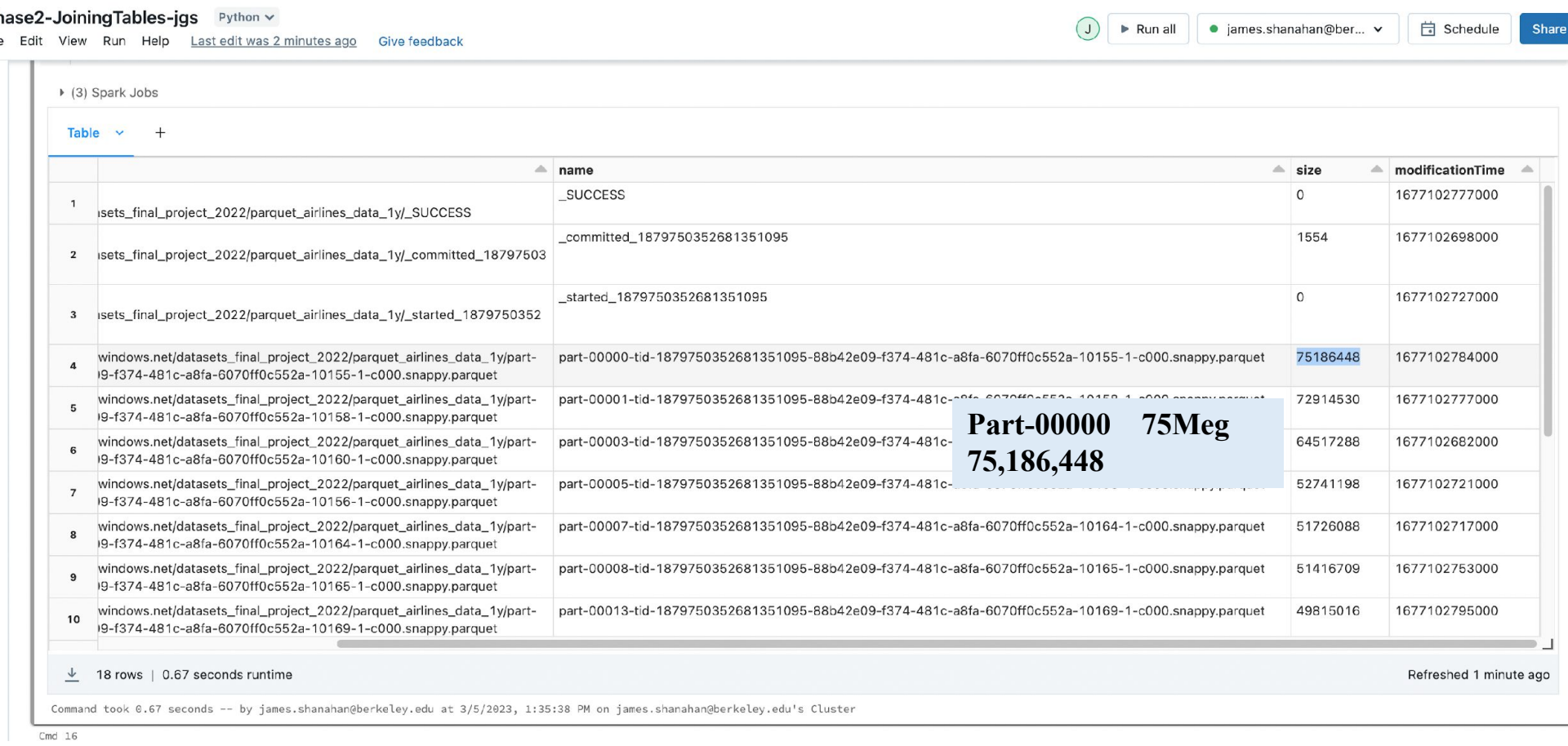
1 row | 52.23 seconds runtime

Table +

	min(DATE)	max(DATE)
1	2015-01-01T00:00:00	2021-12-31T23:59:00

1 row | 52.23 seconds runtime

15 partitions 50M (750 Meg altogether)
1 Gig per year FLIGHTS (parquet compressed)
10 Million flights per year □ 10^6 x (109 cols x 10)
bytes □ 1 GIG (10^9)

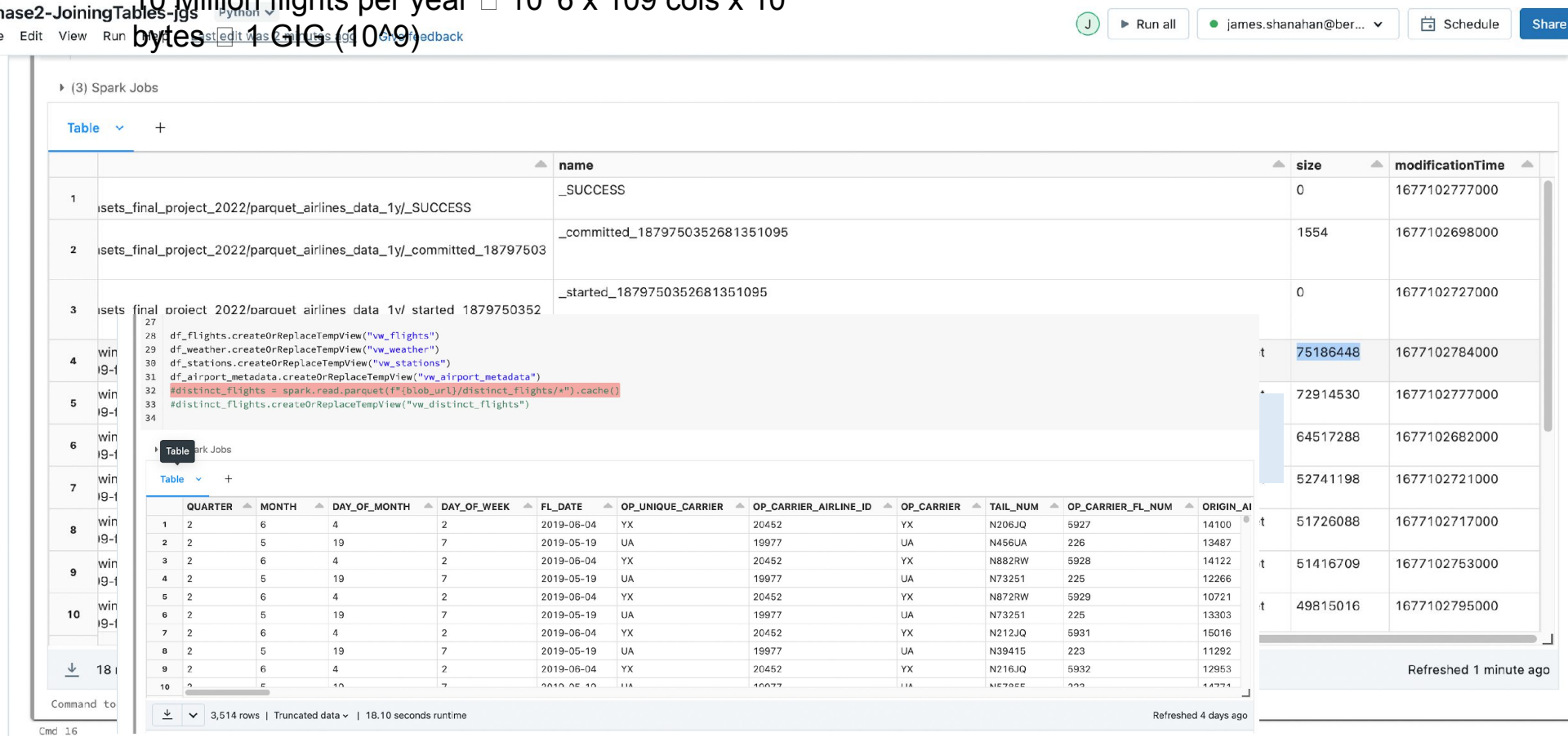


		name	size	modificationTime
1	ısets_final_project_2022/parquet_airlines_data_1y/_SUCCESS	_SUCCESS	0	1677102777000
2	ısets_final_project_2022/parquet_airlines_data_1y/_committed_18797503	_committed_1879750352681351095	1554	1677102698000
3	ısets_final_project_2022/parquet_airlines_data_1y/_started_1879750352	_started_1879750352681351095	0	1677102727000
4	windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-9-f374-481c-a8fa-6070ff0c552a-10155-1-c000.snappy.parquet	part-00000-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10155-1-c000.snappy.parquet	75186448	1677102784000
5	windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-9-f374-481c-a8fa-6070ff0c552a-10158-1-c000.snappy.parquet	part-00001-tid-1879750352681351095-88b42e09-f374-481c-	72914530	1677102777000
6	windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-9-f374-481c-a8fa-6070ff0c552a-10160-1-c000.snappy.parquet	part-00003-tid-1879750352681351095-88b42e09-f374-481c-	64517288	1677102682000
7	windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-9-f374-481c-a8fa-6070ff0c552a-10156-1-c000.snappy.parquet	part-00005-tid-1879750352681351095-88b42e09-f374-481c-	52741198	1677102721000
8	windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-9-f374-481c-a8fa-6070ff0c552a-10164-1-c000.snappy.parquet	part-00007-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10164-1-c000.snappy.parquet	51726088	1677102717000
9	windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-9-f374-481c-a8fa-6070ff0c552a-10165-1-c000.snappy.parquet	part-00008-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10165-1-c000.snappy.parquet	51416709	1677102753000
10	windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-9-f374-481c-a8fa-6070ff0c552a-10169-1-c000.snappy.parquet	part-00013-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10169-1-c000.snappy.parquet	49815016	1677102795000

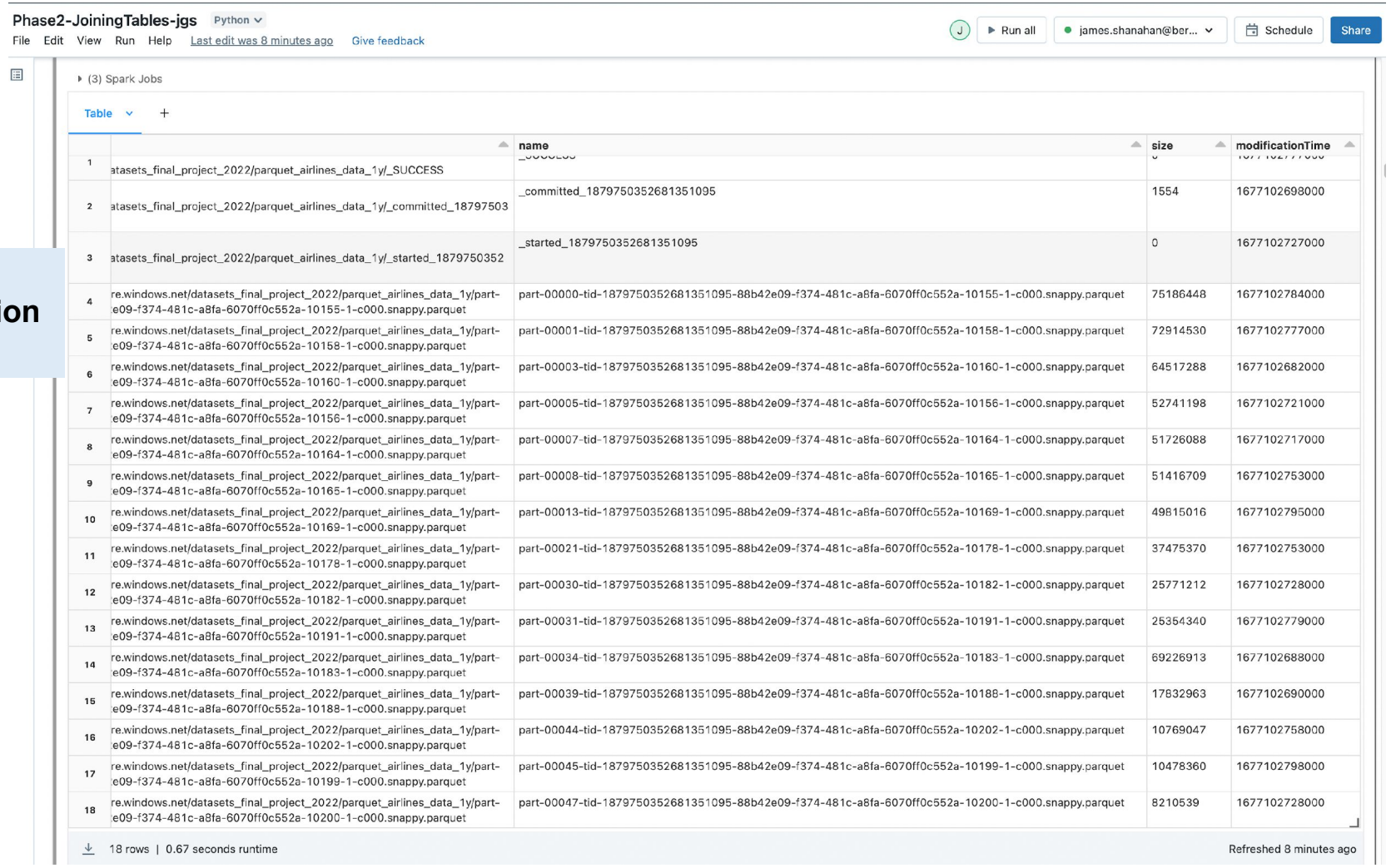
Flights

15 partitions 50M (750 Meg altogether of compressed data)

10 Million flights per year □ 10^6 x 109 cols x 10 bytes = 1 GIG (10^9)



15 partitions



Phase2-JoiningTables-jgs Python

File Edit View Run Help Last edit was 8 minutes ago Give feedback

Run all | james.shanahan@ber... | Schedule | Share

(3) Spark Jobs

Table +

		name	size	modificationTime
1	atasets_final_project_2022/parquet_airlines_data_1y/_SUCCESS			
2	atasets_final_project_2022/parquet_airlines_data_1y/_committed_18797503	_committed_1879750352681351095	1554	1677102698000
3	atasets_final_project_2022/parquet_airlines_data_1y/_started_1879750352	_started_1879750352681351095	0	1677102727000
4	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10155-1-c000.snappy.parquet	part-00000-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10155-1-c000.snappy.parquet	75186448	1677102784000
5	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10158-1-c000.snappy.parquet	part-00001-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10158-1-c000.snappy.parquet	72914530	1677102777000
6	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10160-1-c000.snappy.parquet	part-00003-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10160-1-c000.snappy.parquet	64517288	1677102682000
7	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10156-1-c000.snappy.parquet	part-00005-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10156-1-c000.snappy.parquet	52741198	1677102721000
8	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10164-1-c000.snappy.parquet	part-00007-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10164-1-c000.snappy.parquet	51726088	1677102717000
9	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10165-1-c000.snappy.parquet	part-00008-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10165-1-c000.snappy.parquet	51416709	1677102753000
10	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10169-1-c000.snappy.parquet	part-00013-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10169-1-c000.snappy.parquet	49815016	1677102795000
11	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10178-1-c000.snappy.parquet	part-00021-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10178-1-c000.snappy.parquet	37475370	1677102753000
12	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10182-1-c000.snappy.parquet	part-00030-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10182-1-c000.snappy.parquet	25771212	1677102728000
13	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10191-1-c000.snappy.parquet	part-00031-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10191-1-c000.snappy.parquet	25354340	1677102779000
14	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10183-1-c000.snappy.parquet	part-00034-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10183-1-c000.snappy.parquet	69226913	1677102688000
15	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10188-1-c000.snappy.parquet	part-00039-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10188-1-c000.snappy.parquet	17832963	1677102690000
16	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10202-1-c000.snappy.parquet	part-00044-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10202-1-c000.snappy.parquet	10769047	1677102758000
17	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10199-1-c000.snappy.parquet	part-00045-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10199-1-c000.snappy.parquet	10478360	1677102798000
18	re.windows.net/datasets_final_project_2022/parquet_airlines_data_1y/part-e09-f374-481c-a8fa-6070ff0c552a-10200-1-c000.snappy.parquet	part-00047-tid-1879750352681351095-88b42e09-f374-481c-a8fa-6070ff0c552a-10200-1-c000.snappy.parquet	8210539	1677102728000

18 rows | 0.67 seconds runtime

Refreshed 8 minutes ago

Weather: Monthly
10 Million records of 124 values
10 Gig (compressed)
840 Gig of data (1 TB of data)

Cmd 17

```
1
2  df_weather_3m = spark.read.parquet(f"{blob_url}/datasets_final_project_2022/parquet_weather_data_3m")  # blob_url # https://jshanahan.blob.core.windowsb.net/jgs
3  display(df_weather_3m)
4  print(f"The airport codes table has {df_weather_3m.count():,} rows, {len(df_weather_3m.columns)}")
```

Python

▸ (4) Spark Jobs

▸ df_weather_3m: pyspark.sql.dataframe.DataFrame = [STATION: string, DATE: string ... 122 more fields]

Table +

	STATION	DATE	LATITUDE	LONGITUDE	ELEVATION	NAME	REPORT_TYPE	SOURCE	HourlyAltimeterSetting	HourlyDewPointTemperature
1	52652099999	2015-01-01T02:00:00	39.0833333	100.2833333	1462.0	ZHANGYE, CH	FM-12	4	null	-1
2	52652099999	2015-01-01T05:00:00	39.0833333	100.2833333	1462.0	ZHANGYE, CH	FM-12	4	null	-1
3	52652099999	2015-01-01T08:00:00	39.0833333	100.2833333	1462.0	ZHANGYE, CH	FM-12	4	null	-7
4	52652099999	2015-01-01T11:00:00	39.0833333	100.2833333	1462.0	ZHANGYE, CH	FM-12	4	null	4
5	52652099999	2015-01-01T14:00:00	39.0833333	100.2833333	1462.0	ZHANGYE, CH	FM-12	4	null	7
6	52652099999	2015-01-01T17:00:00	39.0833333	100.2833333	1462.0	ZHANGYE, CH	FM-12	4	null	7
7	52652099999	2015-01-01T20:00:00	39.0833333	100.2833333	1462.0	ZHANGYE, CH	FM-12	4	null	2

2,668 rows | Truncated data | 11.57 seconds runtime — Refreshed 1 minute ago

```
The airport codes table has 30,528,602 rows, 124
Command took 11.57 seconds -- by james.shanahan@berkeley.edu at 3/5/2023, 1:52:49 PM on james.shanahan@berkeley.edu's Cluster
```

Cmd 18

Approximately 23,117,906 weather records for 389 airports (24 Gig of data)

- In the US and its Territories, there are 389 airports.
- We have 7 years of weather data where we record/summarize once every hour.
- This would lead to an estimated 23,117,906 weather records (365 * 7 * 24 * 389 = 23,117,906) of interest;
- SO…….we can discard the the other 870 Million records.

Airport meta data file 57,421 x 12 rows

Airport codes Table Here you will need to import an external airport code conversion set (source: https://datahub.io/core/airport-codes) and join the airport codes to the airline's flights table on the IATA code (3-letter code used by passengers)

The airport codes may refer to either:

- IATA airport code (`iata_code` in the table below), a three-letter code which is used in passenger reservation, ticketing and baggage-handling systems (used in airline flights dataset),
- or ICAO airport code (`gps_code` in the table below) which is a four letter code used by ATC systems and for airports that do not have an IATA airport code (from wikipedia).
- coordinates refers to the lattitude and longitude

Refs:

- https://datahub.io/core/airport-codes
- https://digitalcampus.instructure.com/courses/4868/pages/mids-w261-final-project-dataset-and-cluster?module_item_id=686693

The airport codes may refer to either IATA airport code, a three-letter code which is used in passenger reservation, ticketing and baggage-handling systems, or the ICAO airport code which is a four letter code used by ATC systems and for airports that do not have an IATA airport code (from wikipedia).

ident	type	name	elevation_ft	continent	iso_country	iso_region	municipality	gps_code	iata_code	local_code	coordinates
KSFO	large_airport	San Francisco International Airport	13	NA	US	US-CA	San Francisco	KSFO	SFO	SFO	-122.375, 37.61899948120117

```python
# Airport meta data
print(blob_url)
# Load the airport codes file from the blob storage
df_airport_codes = spark.read.csv(f"{blob_url}/airport-codes_csv.csv", header=True)  # blob_url # wasbs://jgs@jshanahan.blob.core.windows.net
display(df_airport_codes)
print(f"The airport codes table has {df_airport_codes.count():,} rows, {len(df_airport_codes.columns)}")
```

(4) Spark Jobs

df_airport_codes: pyspark.sql.dataframe.DataFrame = [ident: string, type: string ... 10 more fields]

```
wasbs://261-jgs-sandbox@261data.blob.core.windows.net
```

Table +

	ident	type	name	elevation_ft	continent	iso_country	iso_region	municipality	gps_code
1	00A	heliport	Total Rf Heliport	11	NA	US	US-PA	Bensalem	00A
2	00AA	small_airport	Aero B Ranch Airport	3435	NA	US	US-KS	Leoti	00AA
3	00AK	small_airport	Lowell Field	450	NA	US	US-AK	Anchor Point	00AK
4	00AL	small_airport	Epps Airpark	820	NA	US	US-AL	Harvest	00AL
5	00AR	closed	Newport Hospital & Clinic Heliport	237	NA	US	US-AR	Newport	null
6	00AS	small_airport	Fulton Airport	1100	NA	US	US-OK	Alex	00AS
7	00AZ	small_airport	Cordes Airport	3810	NA	US	US-AZ	Cordes	00AZ

10,000 rows | Truncated data | 3.11 seconds runtime — Refreshed now

```
The airport codes table has 57,421 rows, 12
```

Command took 3.11 seconds -- by james.shanahan@berkeley.edu at 3/5/2023, 2:03:45 PM on james.shanahan@berkeley.edu's Cluster

Reporting Carrier On-Time Performance (1987-present)

Flights Data

- Database Name: Airline On-Time Performance Data
  - https://www.transtats.bts.gov/Tables.asp?QO_VQ=EFD&QO_anzr=Nv4yv0r%FDb0-gvzr%FDcr4s14zn0pr%FDQn6n&QO_fu146_anzr=b0-gvzr
- Reporting carriers are required to (or voluntarily) report on-time data for flights they operate: on-time arrival and departure data for non-stop domestic flights by month and year, by carrier and by origin and destination airport. Includes scheduled and actual departure and arrival times, canceled and diverted flights, taxi-out and taxi-in times, causes of delay and cancellation, air time, and non-stop distance.
  - Use Download for individual flight data.
    - https://www.transtats.bts.gov/DL_SelectFields.aspx?gnoyr_VQ=FGJ&QO_fu146_anzr=b0-gvzr
- DAILY: 15-20,000 commercial passenger domestic US flights out of 87,000 flights
  - Commercial makes up 35% of all flights
  - Private planes (30%)
  - Air taxi flights (planes for hire),
  - 5,260 military flights and
  - 2,148 air cargo flights (Federal Express, UPS, etc.)
  - Src: https://www.answerbag.com/q_view/34532

Flights Data Stats

- 20,000 domestic US passenger flights per day
- 500,000 flights per month
  - 109 fields per flight (keep 63) x 8 bytes = 1k per record
  - 500 Meg per month; compressed 50 Meg
  - 25 Meg for 63 fields
- Download data
  - Download URL(by month)
    - https://www.transtats.bts.gov/DL_SelectFields.aspx?gnoyr_VQ=FGJ&QO_fu146_anzr=b0-gvzr

UA 570: has many different origin airports and destination airports. BOS -> EWR > LAX

- UA 570: has many different origin airports and destination airports: BOS □ EWR □ LAX
- 58 flights in Jan 2022; some are missing tail numbers
- Serviced by different Tail Numbers
- Unique flight: Reporting_Airline x Flight_Number_Reporting_Airline X Origin X Dest
  - E.g., flight. UA 570 BOS EWR

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	Year	Quarter	Month	DayofMo	DayOfWe	FlightDat	Reporting_Airline	DOT_ID_F	IATA_CO	Tail_Num	Flight_Nu	OriginAir	OriginAir	OriginCit	Origin	OriginCit	OriginSta	OriginSta	OriginSta	OriginWa	DestAirpo	DestAirpo	DestCityN	Dest	DestCity
67296	2022	1	1	31	1	1/31/22	UA	19977	UA	N54241	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
67297	2022	1	1	31	1	1/31/22	UA	19977	UA	N12003	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
68690	2022	1	1	30	7	1/30/22	UA	19977	UA		570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
68691	2022	1	1	30	7	1/30/22	UA	19977	UA	N12012	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
69962	2022	1	1	29	6	1/29/22	UA	19977	UA		570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
71419	2022	1	1	28	5	1/28/22	UA	19977	UA	N33289	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
71420	2022	1	1	28	5	1/28/22	UA	19977	UA	N12012	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
72899	2022	1	1	27	4	1/27/22	UA	19977	UA	N14731	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
72900	2022	1	1	27	4	1/27/22	UA	19977	UA	N13013	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
73932	2022	1	1	12	3	1/12/22	UA	19977	UA	N62849	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
73933	2022	1	1	12	3	1/12/22	UA	19977	UA	N12003	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
75493	2022	1	1	11	2	1/11/22	UA	19977	UA	N69829	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
75494	2022	1	1	11	2	1/11/22	UA	19977	UA	N91007	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
77108	2022	1	1	10	1	1/10/22	UA	19977	UA	N75429	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
77109	2022	1	1	10	1	1/10/22	UA	19977	UA	N12006	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
78629	2022	1	1	9	7	1/9/22	UA	19977	UA	N462UA	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
78630	2022	1	1	9	7	1/9/22	UA	19977	UA	N14011	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
81225	2022	1	1	26	3	1/26/22	UA	19977	UA	N13720	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
81226	2022	1	1	26	3	1/26/22	UA	19977	UA	N12006	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
82663	2022	1	1	25	2	1/25/22	UA	19977	UA	N24706	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
82664	2022	1	1	25	2	1/25/22	UA	19977	UA	N17002	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
84142	2022	1	1	24	1	1/24/22	UA	19977	UA	N61898	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
84143	2022	1	1	24	1	1/24/22	UA	19977	UA	N17002	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
85528	2022	1	1	23	7	1/23/22	UA	19977	UA	N16709	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
85529	2022	1	1	23	7	1/23/22	UA	19977	UA	N12006	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
86818	2022	1	1	8	6	1/8/22	UA	19977	UA	N69020	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
88407	2022	1	1	7	5	1/7/22	UA	19977	UA	N69833	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
88408	2022	1	1	7	5	1/7/22	UA	19977	UA	N12003	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
90026	2022	1	1	6	4	1/6/22	UA	19977	UA	N62884	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark
90027	2022	1	1	6	4	1/6/22	UA	19977	UA	N91007	570	11618	1161802	31703	EWR	Newark, NJ	NJ	34	New Jersey	21	12892	1289208	32575	LAX	Los Ang
91596	2022	1	1	5	3	1/5/22	UA	19977	UA	N37474	570	10721	1072102	30721	BOS	Boston, MA	MA	25	Massachusel	13	11618	1161802	31703	EWR	Newark

On Time Reporting Carrier On Ti

Ready 58 of 537902 records found Accessibility: Investigate 100%

OP_CARRIER_FL_NUM (aka Flight_Number_Reporting_Airline)is 1898 Flight UA 1898 DEN DFW

- Unique flight: using ATP column names
  - Reporting_Airline X Flight_Number_Reporting_Airline X Origin X Dest

- 2 4 8 3 2020-04-08 00:00:00 UA 19977 UA N469UA 1898 11292 11292 02 30325 DEN Denver, CO CO 8 Colorado 82 11298 11298 06 30194 DFW Dallas/Fort Worth, TX TX 48 Texas 74 949 941 -8 0 0 -1 0900-0959 13 954 1216 10 1248 1226 -22 0 0 -2 1200-1259 0 null 0 119 105 82 1 641 3 null null null null null null null null 2020

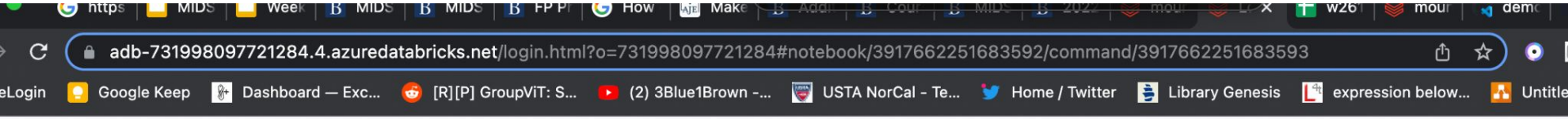
Databricks

Databricks develops a web-based platform for working with Spark, that provides automated cluster management and IPython-style notebooks.

Databricks grew out of the AMPLab project at University of California, Berkeley that was involved in making Apache Spark, an open-source distributed computing framework built atop Scala.

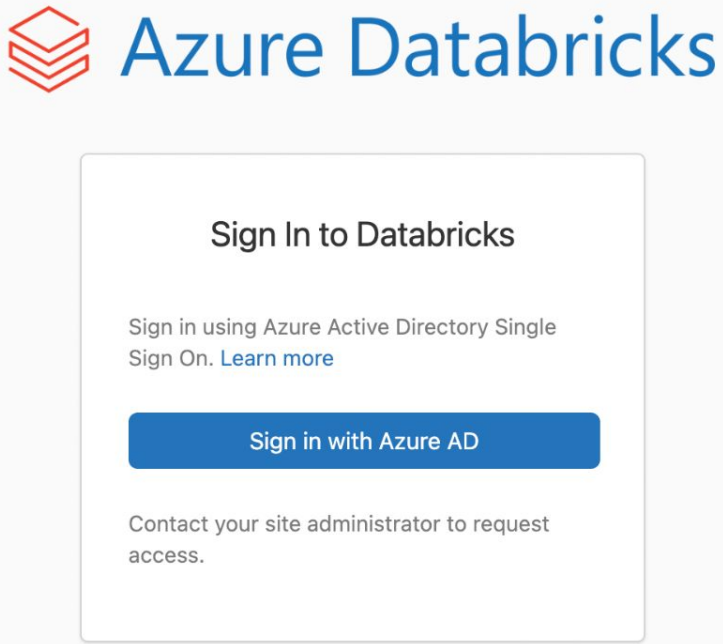
Databricks develops and sells a cloud data platform using the marketing term "lakehouse", a portmanteau based on the terms "data warehouse" and "data lake".[23] Databricks' lakehouse is based on the open source Apache Spark framework that allows analytical queries against semi-structured data without a traditional database schema.[24] In October, 2022, Lakehouse received FedRAMP authorized status for use with the U.S. federal government and contractors.

Databricks account



Azure active directory (AAD) invitations have been sent

And your project team Databricks workspaces are currently been setup

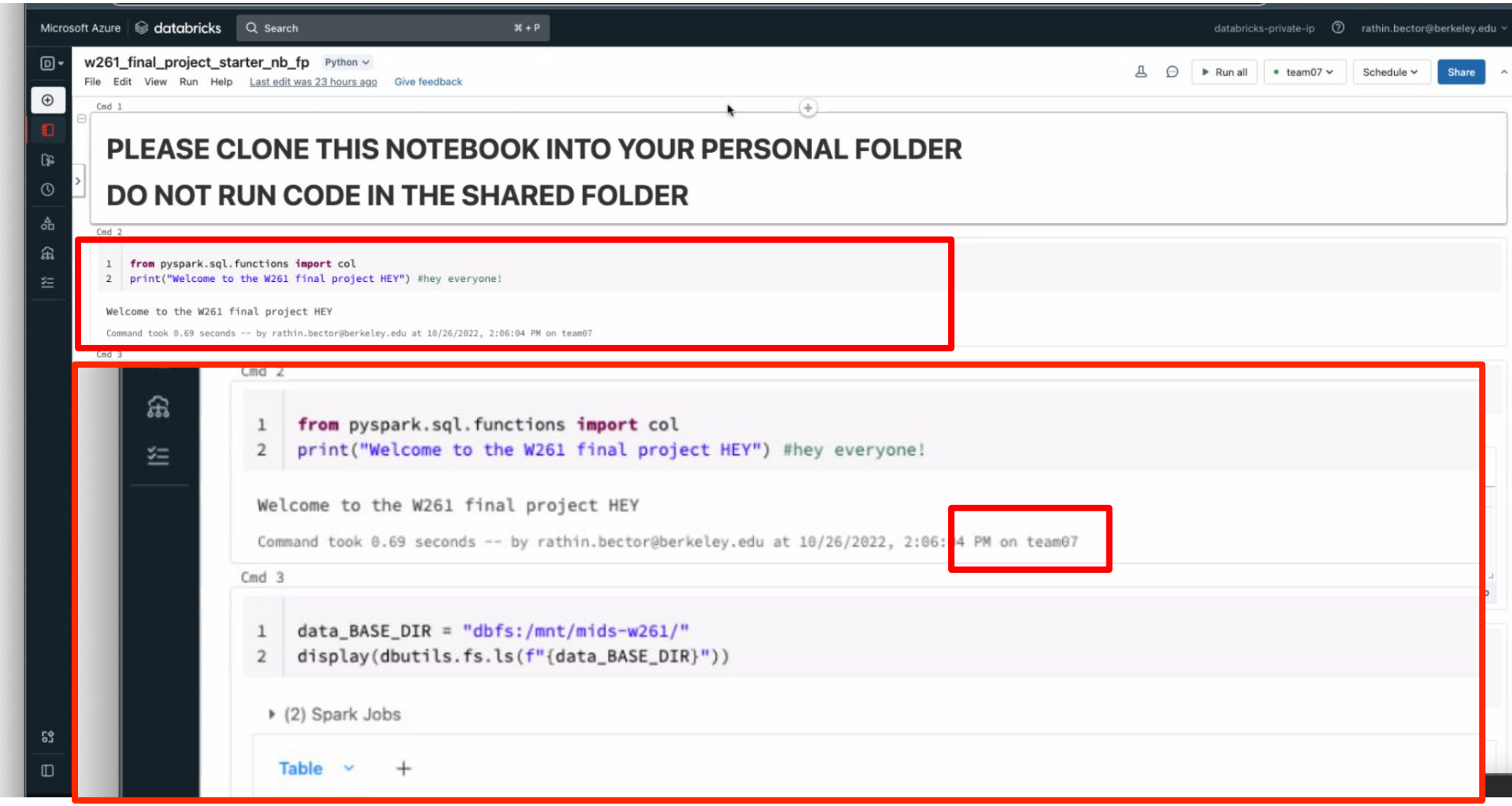


Databricks Notebooks

- Local notebook
  - https://www.dropbox.com/s/y6vxpdoz34bruog/demo10_workbook.ipynb?dl=0

- Starter notebooks on Databricks
  - Click to access the Final Project Starter notebook:
    - Final project starter notebook
    - And attach your notebook to your Project Team's cluster Databricks (available in US-WEST 2 Region) to run the cells in your notebook on a Spark Cluster (that auto-scales from 2 nodes to 4 nodes for the early phases of the project).
  - For more background on Blob Storage, please see this notebook: Blob storage Notebook.,
    - Details on setting up an account and blob storage and on checkpointing your data
  -

Attach notebook to your team's cluster (team 7 here for Rathin)



Cmd 31

Python

```
mids261_mount_path      = "/mnt/mids-w261" #no permission to write to this mounted blob storage;
#paths =[f"{blob_url}/OTPW_60M"] #a list of paths
paths =[f"{mids261_mount_path}/OTPW_60M"] #a list of paths
OTPW_5_years = spark.read.option("header",True) \
                        .option("compression", "gzip") \
                        .option("delimiter",",") \
                        .csv(paths).cache()
OTPW_5_years.createOrReplaceTempView("OTPW_5_years")
print(f"The OTPW_5_years (flights joined with weather) table has {OTPW_5_years.count():,} rows, {len(OTPW_5_years.columns)} cols")

display(OTPW_5_years.limit(100))

```

▸ (4) Spark Jobs

▸ OTPW_5_years_v2: pyspark.sql.dataframe.DataFrame = [QUARTER: string, DAY_OF_MONTH: string ... 212 more fields]

```
The OTPW_5_years_v2 (flights joined with weather) table has 31,673,119 rows, 214 cols
```

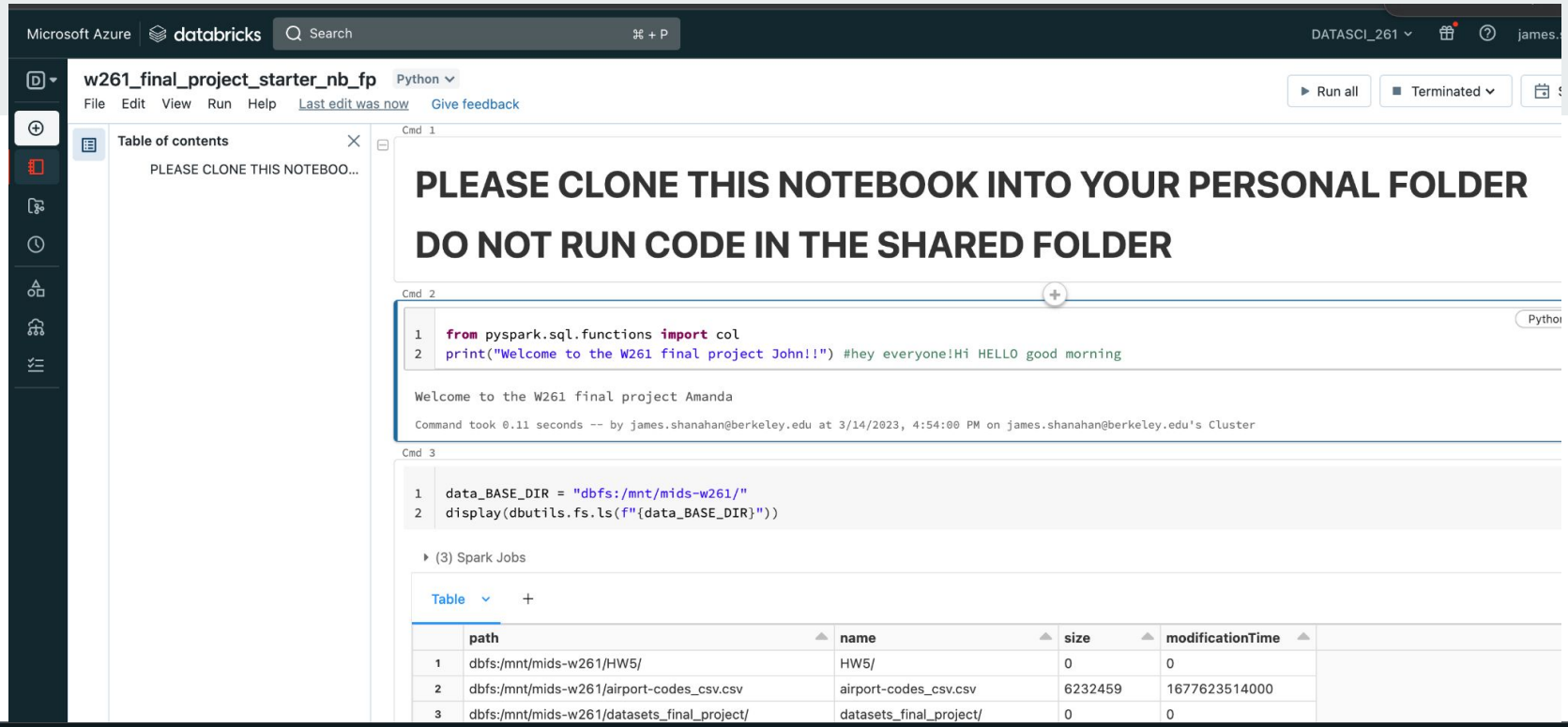
Table +

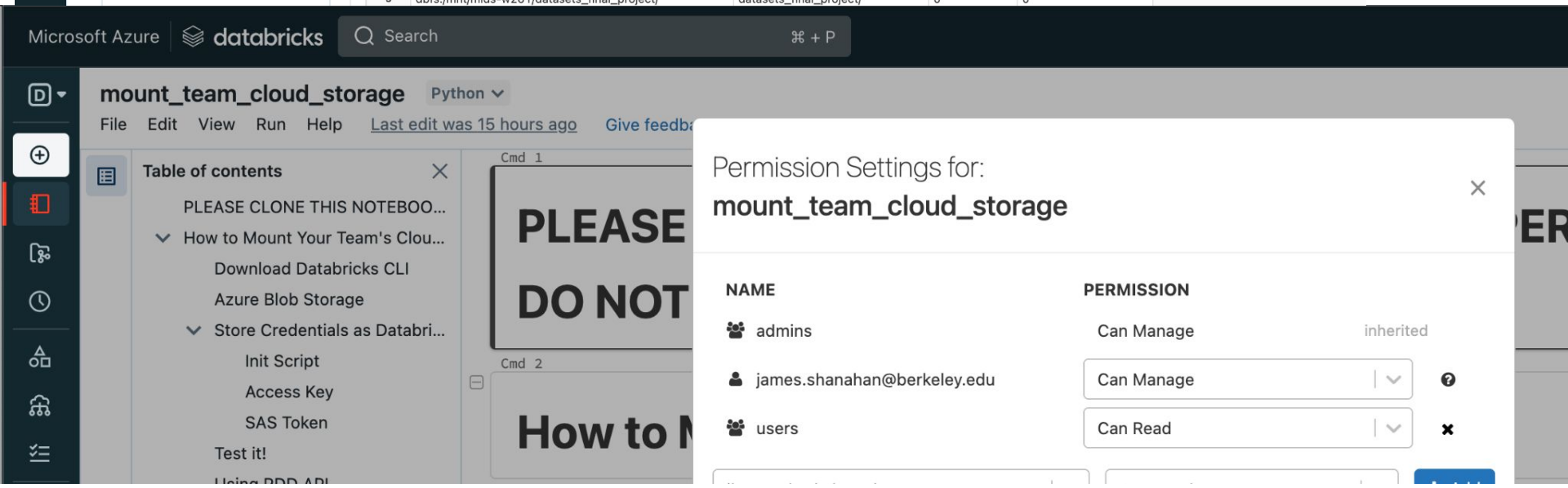
	QUARTER	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	OP_UNIQUE_CARRIER	OP_CARRIER_AIRLINE_ID	OP_CARRIER	TAIL_NUM	OP_CARRIER_FL_NUM	ORIGIN_A
1	2	24	2	2018-04-24	9E	20363	9E	N391CA	3280	11618
2	2	6	7	2018-05-06	9E	20363	9E	N8877A	3280	11433
3	4	17	3	2018-10-17	9E	20363	9E	N8976E	3280	10397
4	1	18	5	2019-01-18	9E	20363	9E	N8683B	3280	11150
5	1	14	4	2019-03-14	9E	20363	9E	N8974C	3280	10397
6	2	27	1	2019-05-27	9E	20363	9E	N298PQ	3280	14122
	4	6	7	2019-10-06	9E	20363	9E	N8946A	3280	11193

100 rows | 11.42 minutes runtime

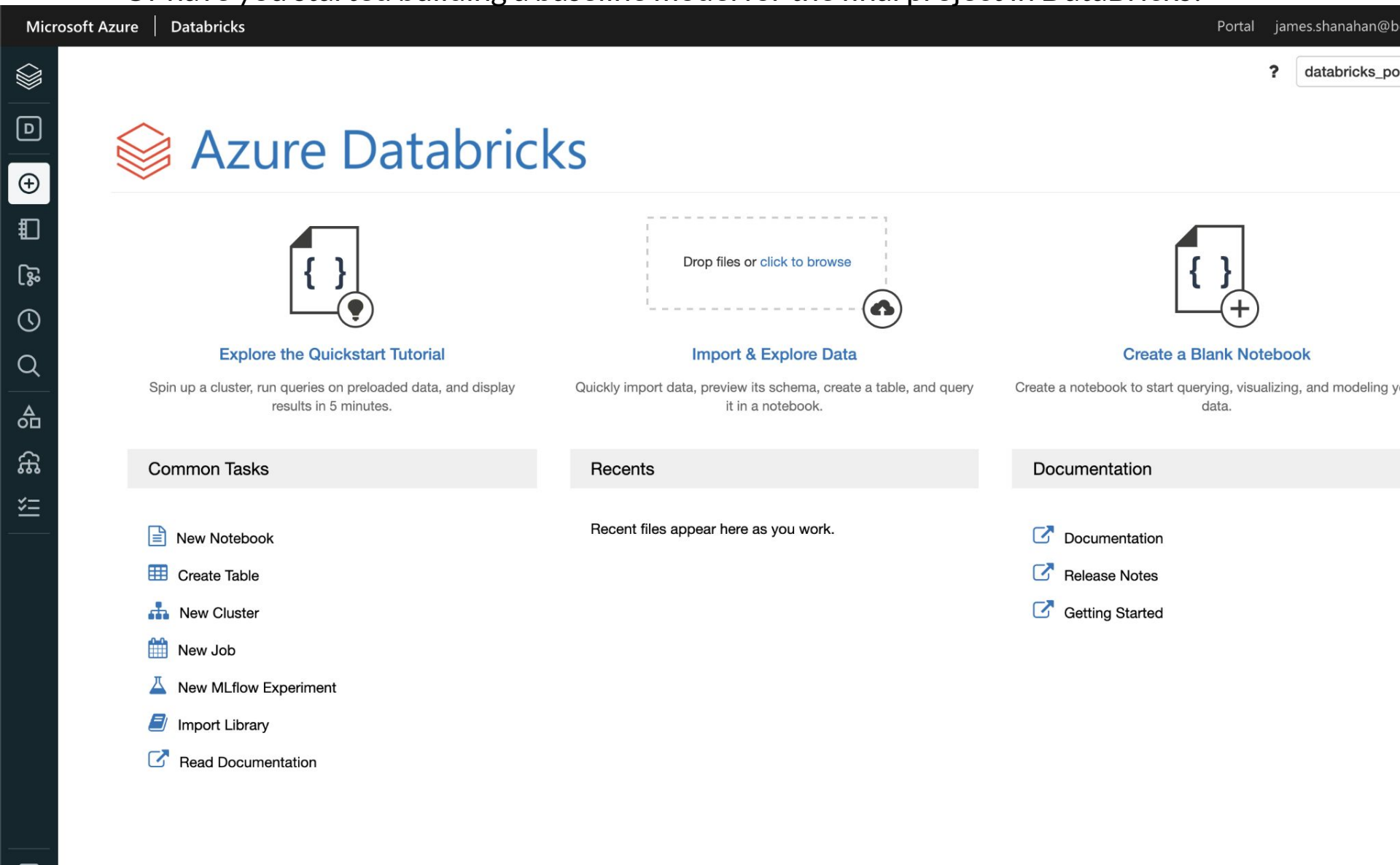
Refreshed 12 minutes ago

```
Command took 11.42 minutes -- by james.shanahan@berkeley.edu at 4/14/2023, 1:31:20 PM on Jimi's private
```


Microsoft Azure
databricks
Search
⌘ + P
DATASCI_261
w261_final_project_starter_nb_fp
Python
File Edit View Run Help Last edit was now Give feedback
Run all
Terminated
Table of contents
PLEASE CLONE THIS NOTEBOO...
Cmd 1
PLEASE CLONE THIS NOTEBOOK INTO YOUR PERSONAL FOLDER
DO NOT RUN CODE IN THE SHARED FOLDER
Cmd 2
from pyspark.sql.functions import col
print("Welcome to the W261 final project John!!") #hey everyone!Hi HELLO good morning
Welcome to the W261 final project Amanda
Command took 0.11 seconds -- by james.shanahan@berkeley.edu at 3/14/2023, 4:54:00 PM on james.shanahan@berkeley.edu's Cluster
Cmd 3
data_BASE_DIR = "dbfs:/mnt/mids-w261/"
display(dbutils.fs.ls(f"{data_BASE_DIR}"))
(3) Spark Jobs
Table
path
name
size
modificationTime
1 dbfs:/mnt/mids-w261/HW5/ HW5/ 0 0
2 dbfs:/mnt/mids-w261/airport-codes_csv.csv airport-codes_csv.csv 6232459 1677623514000
3 dbfs:/mnt/mids-w261/datasets_final_project/ datasets_final_project/ 0 0

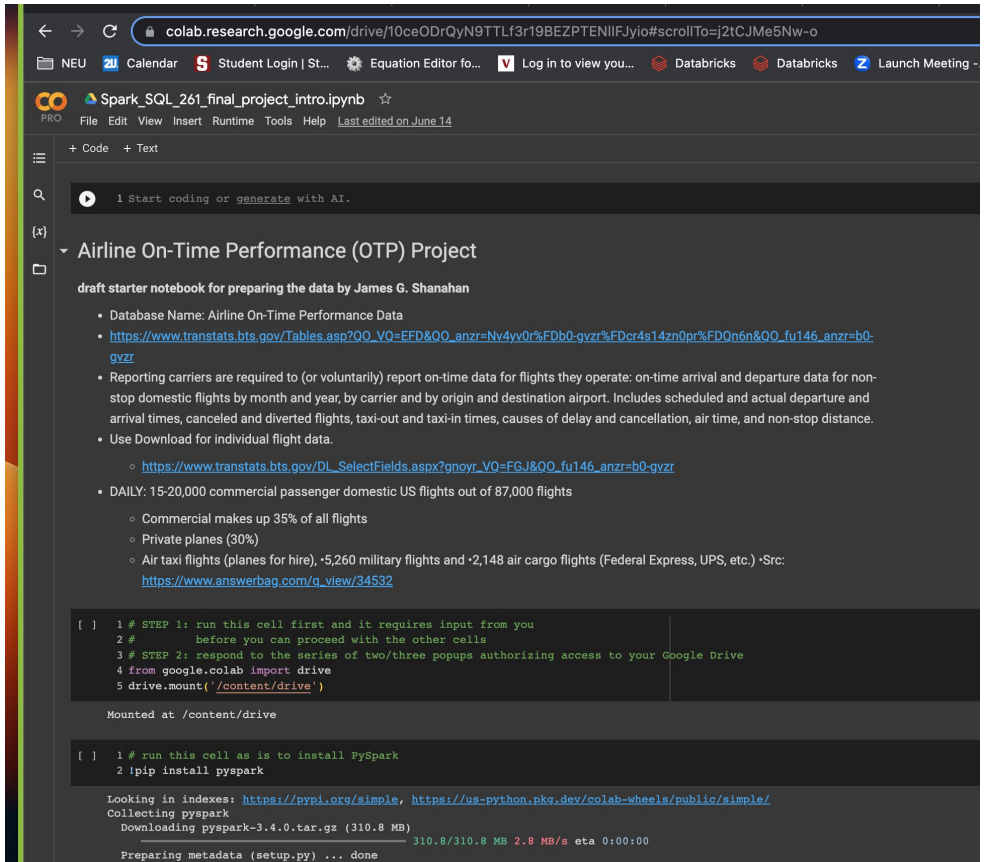

Microsoft Azure
databricks
Search
⌘ + P
mount_team_cloud_storage
Python
File Edit View Run Help Last edit was 15 hours ago
Table of contents
PLEASE CLONE THIS NOTEBOO...
How to Mount Your Team's Clou...
Download Databricks CLI
Azure Blob Storage
Store Credentials as Databri...
Init Script
Access Key
SAS Token
Test it!
Cmd 1
PLEASE
DO NOT
Cmd 2
How to M
Permission Settings for:
mount_team_cloud_storage
NAME
PERMISSION
admins
Can Manage
inherited
james.shanahan@berkeley.edu
Can Manage
users
Can Read

- Has your team logged into DataBricks?
- Or have you started building a baseline model for the final project in DataBricks?



Google Colab with joined data example for one day 1/1/2015

https://colab.research.google.com/drive/10ceODrQyN9TTLf3r19BEZPTENlIFJyio?usp=sharing



Python For Data Science *Cheat Sheet*

PySpark - SQL Basics

Learn Python for data science Interactively at www.DataCamp.com

PySpark & Spark SQL

Spark SQL is Apache Spark's module for working with structured data.

Initializing SparkSession

A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()
```

Creating DataFrames

From RDDs

```
>>> from pyspark.sql.types import *
```

Infer Schema

```
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(","))
>>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))
>>> peopledf = spark.createDataFrame(people)
```

Specify Schema

```
>>> people = parts.map(lambda p: Row(name=p[0],
                                     age=int(p[1].strip())))
>>> schemaString = "name age"
>>> fields = [StructField(field_name, StringType(), True) for
field_name in schemaString.split()]
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
  +---------+---+
  |     name|age|
  +---------+---+
  |     Mine| 28|
  |    Filip| 29|
  |Jonathan | 30|
  +---------+---+
```

From Spark Data Sources

JSON

```
>>> df = spark.read.json("customer.json")
>>> df.show()
 +--------------------+---+---------+--------+--------------------+
 |             address|age|firstName|lastName|         phoneNumber|
 +--------------------+---+---------+--------+--------------------+
 |[New York,10021,N...| 25|     John|   Smith|[[212 555-1234,ho...|
 |[New York,10021,N...| 21|     Jane|     Doe|[[322 888-1234,ho...|
 +--------------------+---+---------+--------+--------------------+
>>> df2 = spark.read.load("people.json", format="json")
```

Parquet files

```
>>> df3 = spark.read.load("users.parquet")
```

TXT files

```
>>> df4 = spark.read.text("people.txt")
```

Duplicate Values

```
>>> df = df.dropDuplicates()
```

Queries

```
>>> from pyspark.sql import functions as F
```

Select

```
>>> df.select("firstName").show()
>>> df.select("firstName","lastName") \
      .show()
>>> df.select("firstName",
              "age",
               explode("phoneNumber") \
              .alias("contactInfo")) \
      .select("contactInfo.type",
              "firstName",
              "age") \
      .show()
>>> df.select(df["firstName"],df["age"]+ 1)
      .show()
>>> df.select(df['age'] > 24).show()
```

Show all entries in `firstName` column

Show all entries in `firstName`, `age` and `type`

Show all entries in `firstName` and `age`, add `1` to the entries of `age`

Show all entries where `age` >24

When

```
>>> df.select("firstName",
              F.when(df.age > 30, 1) \
              .otherwise(0)) \
      .show()
>>> df[df.firstName.isin("Jane","Boris")]
              .collect()
```

Show `firstName` and 0 or 1 depending on `age` >30

Show `firstName` if in the given options

Like

```
>>> df.select("firstName",
              df.lastName.like("Smith")) \
      .show()
```

Show `firstName`, and `lastName` is `TRUE` if `lastName` is like `Smith`

Startswith - Endswith

```
>>> df.select("firstName",
              df.lastName \
                .startswith("Sm")) \
      .show()
>>> df.select(df.lastName.endswith("th")) \
      .show()
```

Show `firstName`, and `TRUE` if `lastName` starts with `Sm`

Show last names ending in `th`

Substring

```
>>> df.select(df.firstName.substr(1, 3) \
                          .alias("name")) \
      .collect()
```

Return substrings of `firstName`

Between

```
>>> df.select(df.age.between(22, 24)) \
      .show()
```

Show `age`: values are `TRUE` if between `22` and `24`

Add, Update & Remove Columns

Adding Columns

```
>>> df = df.withColumn('city',df.address.city) \
           .withColumn('postalCode',df.address.postalCode) \
           .withColumn('state',df.address.state) \
           .withColumn('streetAddress',df.address.streetAddress) \
           .withColumn('telePhoneNumber',
                       explode(df.phoneNumber.number)) \
           .withColumn('telePhoneType',
                       explode(df.phoneNumber.type))
```

Updating Columns

```
>>> df = df.withColumnRenamed('telePhoneNumber', 'phoneNumber')
```

Removing Columns

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.address).drop(df.phoneNumber)
```

GroupBy

```
>>> df.groupBy("age")\
      .count() \
      .show()
```

Group by `age`, count the members in the groups

Filter

```
>>> df.filter(df["age"]>24).show()
```

Filter entries of `age`, only keep those records of which the values are `>24`

Sort

```
>>> peopledf.sort(peopledf.age.desc()).collect()
>>> df.sort("age", ascending=False).collect()
>>> df.orderBy(["age","city"],ascending=[0,1])\
      .collect()
```

Missing & Replacing Values

```
>>> df.na.fill(50).show()
>>> df.na.drop().show()
>>> df.na \
      .replace(10, 20) \
      .show()
```

Replace null values
Return new `df` omitting rows with null values
Return new `df` replacing one value with another

Repartitioning

```
>>> df.repartition(10)\
      .rdd \
      .getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions()
```

`df` with 10 partitions

`df` with 1 partition

Running SQL Queries Programmatically

Registering DataFrames as Views

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")
```

Query Views

```
>>> df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people")\
                 .show()
```

Output

Data Structures

```
>>> rdd1 = df.rdd
>>> df.toJSON().first()
>>> df.toPandas()
```

Convert `df` into an RDD
Convert `df` into a RDD of string
Return the contents of `df` as Pandas DataFrame

Write & Save to Files

```
>>> df.select("firstName", "city")\
      .write \
      .save("nameAndCity.parquet")
>>> df.select("firstName", "age") \
      .write \
      .save("namesAndAges.json",format="json")
```

Stopping SparkSession

```
>>> spark.stop()
```

Inspect Data

```
>>> df.dtypes
>>> df.show()
>>> df.head()
>>> df.first()
>>> df.take(2)
>>> df.schema
```

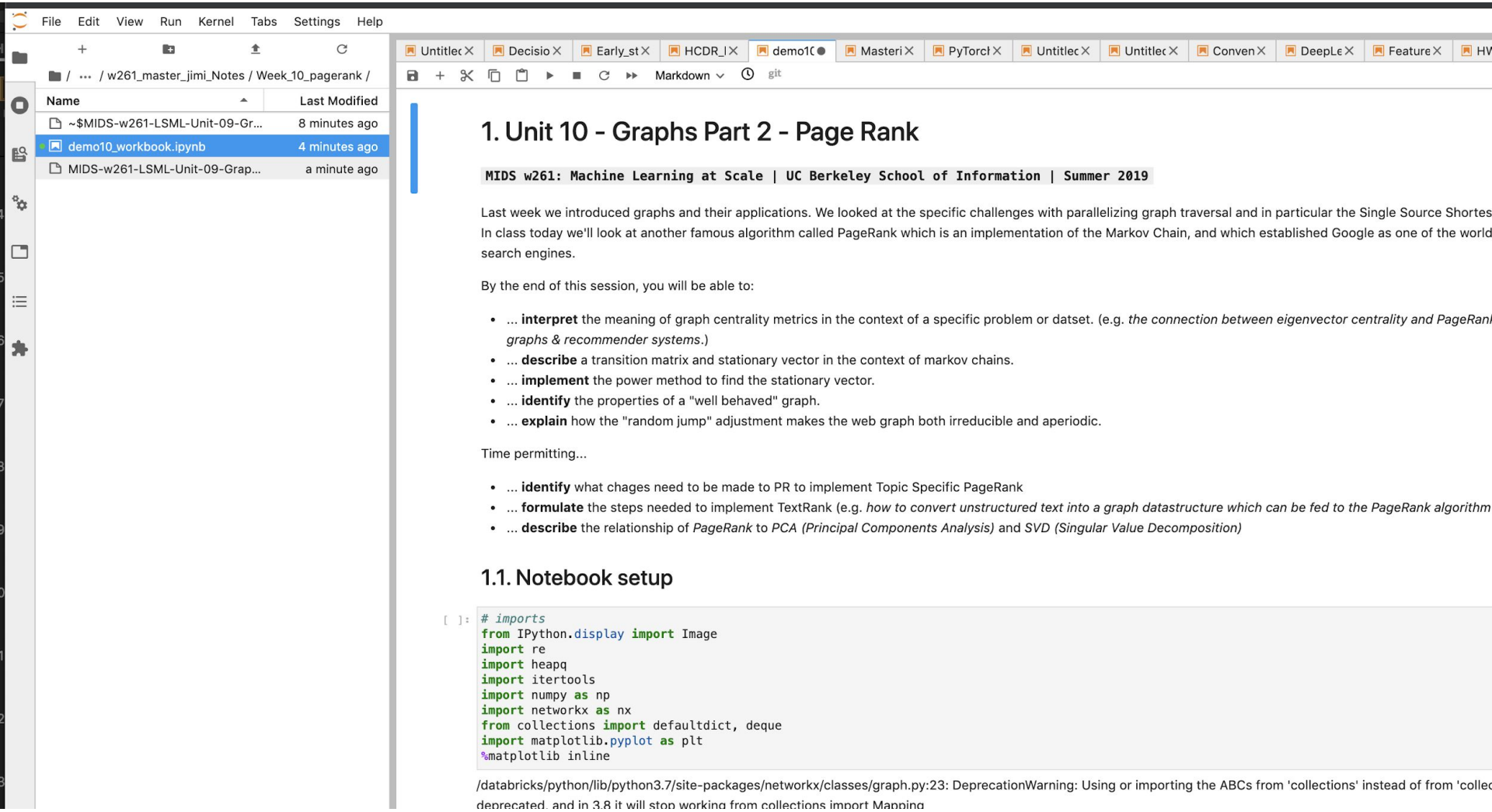
Return `df` column names and data types
Display the content of `df`
Return first `n` rows
Return first row
Return the first `n` rows
Return the schema of `df`

```
>>> df.describe().show()
>>> df.columns
>>> df.count()
>>> df.distinct().count()
>>> df.printSchema()
>>> df.explain()
```

Compute summary statistics
Return the columns of `df`
Count the number of rows in `df`
Count the number of distinct rows in `df`
Print the schema of `df`
Print the (logical and physical) plans

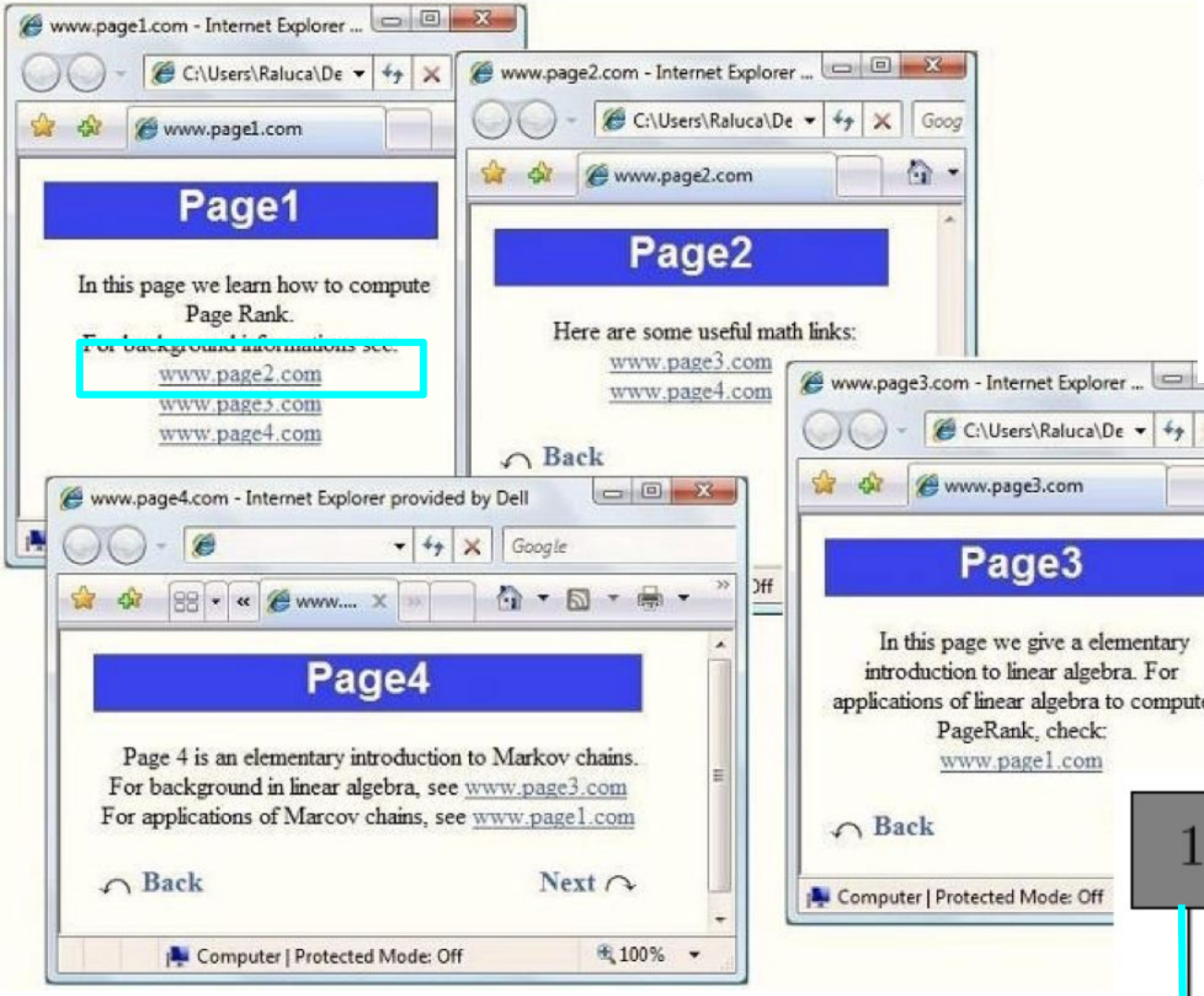
Live Session Outline

- **Page Rank: executive summary**
  - Adjacency Matrices and Markov Processes
  - Calculating Steady State and The Power Method
  - Distributed PageRank
  - Dangling Nodes
  - Topic Specific PageRank
- **Bonus [Optional] Topics**
  - Contextual advertising [Optional]
  - Text as graph: TextRank [Optional]
    - Keyword extraction (from text/target pages)
    - Text Summarization

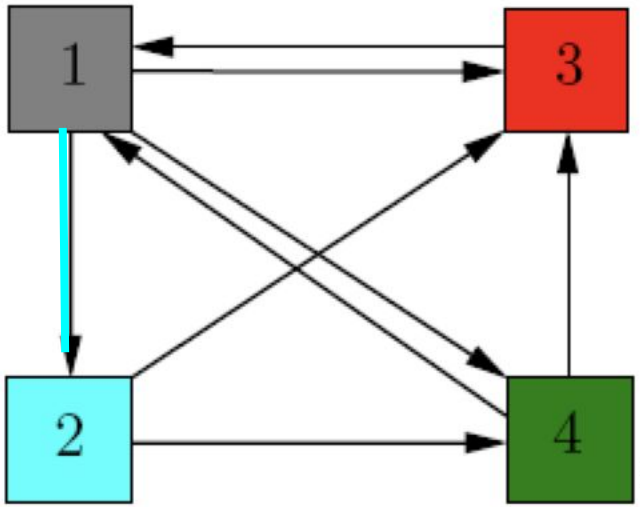

File Edit View Run Kernel Tabs Settings Help
/ w261_master_jimi_Notes / Week_10_pagerank /
Name
Last Modified
~$MIDS-w261-LSML-Unit-09-Gr...
8 minutes ago
demo10_workbook.ipynb
4 minutes ago
MIDS-w261-LSML-Unit-09-Grap...
a minute ago
Markdown
git
1. Unit 10 - Graphs Part 2 - Page Rank
MIDS w261: Machine Learning at Scale | UC Berkeley School of Information | Summer 2019
Last week we introduced graphs and their applications. We looked at the specific challenges with parallelizing graph traversal and in particular the Single Source Shortes
In class today we'll look at another famous algorithm called PageRank which is an implementation of the Markov Chain, and which established Google as one of the world
search engines.
By the end of this session, you will be able to:
... interpret the meaning of graph centrality metrics in the context of a specific problem or datset. (e.g. the connection between eigenvector centrality and PageRank
graphs & recommender systems.)
... describe a transition matrix and stationary vector in the context of markov chains.
... implement the power method to find the stationary vector.
... identify the properties of a "well behaved" graph.
... explain how the "random jump" adjustment makes the web graph both irreducible and aperiodic.
Time permitting...
... identify what chages need to be made to PR to implement Topic Specific PageRank
... formulate the steps needed to implement TextRank (e.g. how to convert unstructured text into a graph datastructure which can be fed to the PageRank algorithm
... describe the relationship of PageRank to PCA (Principal Components Analysis) and SVD (Singular Value Decomposition)
1.1. Notebook setup
[ ]:
# imports
from IPython.display import Image
import re
import heapq
import itertools
import numpy as np
import networkx as nx
from collections import defaultdict, deque
import matplotlib.pyplot as plt
%matplotlib inline
/databricks/python/lib/python3.7/site-packages/networkx/classes/graph.py:23: DeprecationWarning: Using or importing the ABCs from 'collections' instead of from 'collec
deprecated, and in 3.8 it will stop working from collections import Mapping

Pagerank math

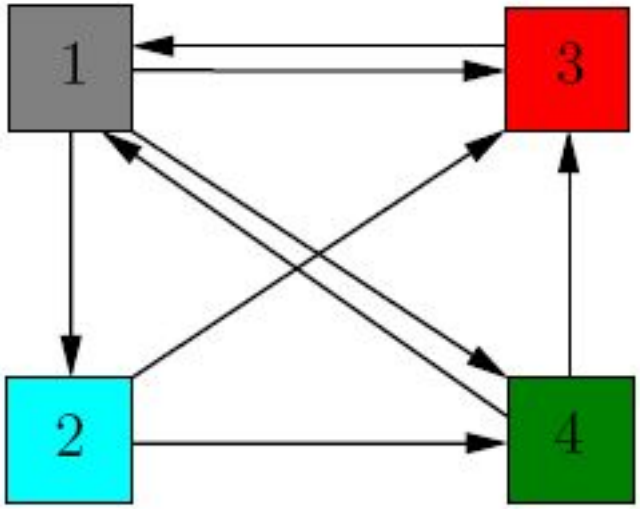
- http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html
- **3 math perspectives [see next slides]**
  - **Dynamical systems point of view**
  - ***Linear algebra point of view***
  - **Probabilistic point of view**

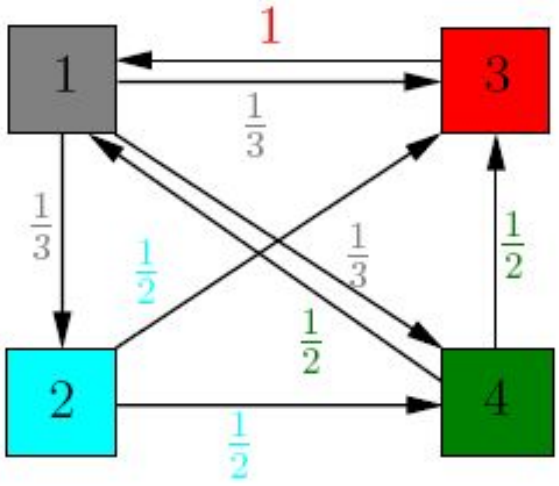


$$A = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$$



http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html





$$A = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$$

When web site i references j, we add a directed edge between node i and node j in the graph.

In our model, each page should transfer evenly its importance to the pages that it links to. Node 1 has 3 outgoing edges, so it will pass on $\frac{1}{3}$ of its importance to each of the other 3 nodes. Node 3 has only one outgoing edge, so it will pass on all of its importance to node 1. In general, if a node has k outgoing edges, it will pass on $\frac{1}{k}$ of its importance to each of the nodes that it links to. Let us better visualize the process by assigning weights to each edge.

Adjacency matrix A, and importance vector

Dynamical systems point of view:

Suppose that initially the importance is uniformly distributed among the 4 nodes, each getting ¼. Denote by v the initial rank vector, having all entries equal to ¼. Each incoming link increases the importance of a web page, so at step 1, we update the rank of each page by adding to the current value the importance of the incoming links. This is the same as multiplying the matrix A with v . At step 1, the new importance vector is $v_1 = Av$. We can iterate the process, thus at step 2, the updated importance vector is $v_2 = A(Av) = A^2v$. Numeric computations give:

$$v = \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix}, \quad Av = \begin{pmatrix} 0.37 \\ 0.08 \\ 0.33 \\ 0.20 \end{pmatrix}, \quad A^2 v = A(Av) = A \begin{pmatrix} 0.37 \\ 0.08 \\ 0.33 \\ 0.20 \end{pmatrix} = \begin{pmatrix} 0.43 \\ 0.12 \\ 0.27 \\ 0.16 \end{pmatrix}$$

$$A = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$$

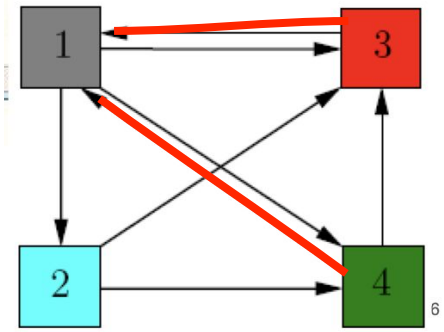
$$A^3 v = \begin{pmatrix} 0.35 \\ 0.14 \\ 0.29 \\ 0.20 \end{pmatrix}, \quad A^4 v = \begin{pmatrix} 0.39 \\ 0.11 \\ 0.29 \\ 0.19 \end{pmatrix}, \quad A^5 v = \begin{pmatrix} 0.39 \\ 0.13 \\ 0.28 \\ 0.19 \end{pmatrix}$$

$$A^6 v = \begin{pmatrix} 0.38 \\ 0.13 \\ 0.29 \\ 0.19 \end{pmatrix}, \quad A^7 v = \begin{pmatrix} 0.38 \\ 0.12 \\ 0.29 \\ 0.19 \end{pmatrix}, \quad A^8 v = \begin{pmatrix} 0.38 \\ 0.12 \\ 0.29 \\ 0.19 \end{pmatrix}$$

We notice that the sequences of iterates v, Av, ..., $A^k v$ tends to the equilibrium value $v^* = \begin{pmatrix} 0.38 \\ 0.12 \\ 0.29 \\ 0.19 \end{pmatrix}$. We call this the PageRank vector of our web graph.

Linear algebra point of view: Eigendecomp

Let us denote by x_1, x_2, x_3, and x_4 the importance of the four pages. Analyzing the situation at each node we get the system:



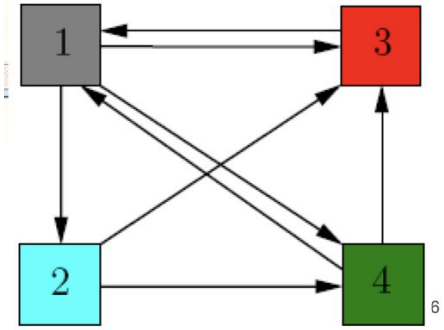
$$\begin{cases} x_1 = 1 \cdot x_3 + \frac{1}{2} \cdot x_4 \\ x_2 = \frac{1}{3} \cdot x_1 \\ x_3 = \frac{1}{3} \cdot x_1 + \frac{1}{2} \cdot x_2 + \frac{1}{2} \cdot x_4 \\ x_4 = \frac{1}{3} \cdot x_1 + \frac{1}{2} \cdot x_2 \end{cases} \qquad A = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$$

This is equivalent to asking for the solutions of the equations $A \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$. From Example 6 in Lecture 1 we know that the eigenvectors corresponding to the eigenvalue 1 are of the form $c \cdot \begin{bmatrix} 12 \\ 4 \\ 9 \\ 6 \end{bmatrix}$. Since PageRank should reflect only the relative importance of the nodes, and since the eigenvectors are just scalar multiples of each other, we can choose any of them to be our PageRank vector. Choose v^* to be the unique eigenvector with the sum of all entries equal to 1. (We will sometimes refer to it as the probabilistic eigenvector corresponding to the eigenvalue 1). The eigenvector $\frac{1}{31} \cdot \begin{bmatrix} 12 \\ 4 \\ 9 \\ 6 \end{bmatrix} \sim \begin{bmatrix} 0.38 \\ 0.12 \\ 0.29 \\ 0.19 \end{bmatrix}$ is our PageRank vector.

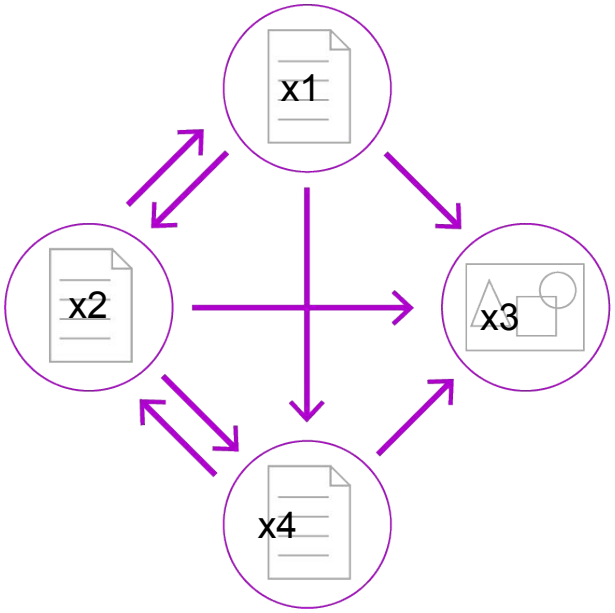
Probabilistic POV: Random surfer, random walk stationary distribution (similar to dynamical systems)

Probabilistic point of view:

Since the importance of a web page is measured by its popularity (how many incoming links it has), we can view the importance of page i as the probability that a random surfer on the Internet that opens a browser to any page and starts following hyperlinks, visits the page i. We can interpret the weights we assigned to the edges of the graph in a probabilistic way: A random surfer that is currently viewing web page 2, has ½ probability to go to page 3, and ½ probability to go to page 4. We can model the process as a random walk on graphs. Each page has equal probability ¼ to be chosen as a starting point. So, the initial probability distribution is given by the column vector $[¼\ ¼\ ¼\ ¼]^t$. The probability that page i will be visited after one step is equal to Ax, and so on. The probability that page i will be visited after k steps is equal to $A^k x$. The sequence Ax, A^2x, A^3x, ..., A^kx, ... converges in this case to a unique probabilistic vector v^*. In this context v^* is called the stationary distribution and it will be our Page Rank vector. Moreover, the i^{th} entry in the vector v^* is simply the probability that at each moment a random surfer visits page i. The computations are identical to the ones we did in the dynamical systems interpretation, only the meaning we attribute to each step being slightly different.



$$A = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$$



Teleportation adjustments for PageRank

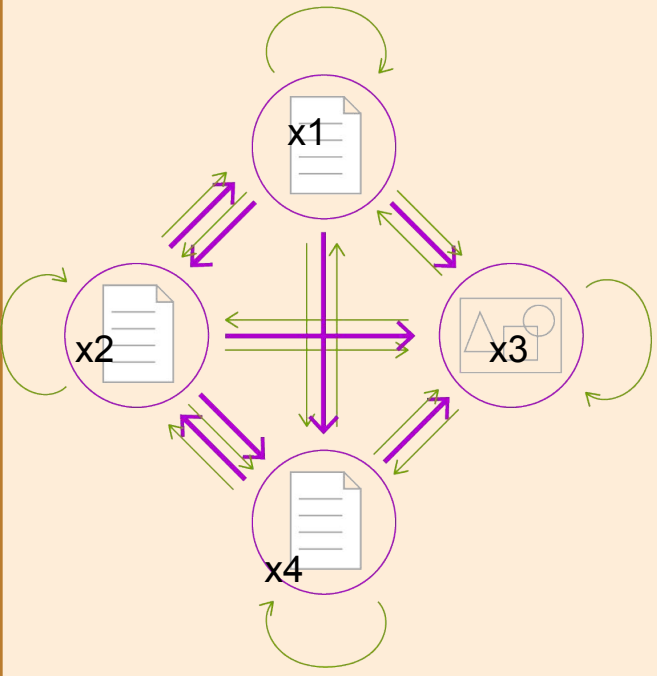
Random Jump factor adds a small amount of probability to each node to teleport to any other node.

Both issues of periodicity and irreducibility are solved at once.

1: Irreducibility/Reachable A Markov chain is said to be **irreducible** if its state space is a single communicating class; in other words, if it is possible to get to any state from any state.

2: Aperiodic: A Markov chain is aperiodic if every state is aperiodic.
An irreducible Markov chain only needs one aperiodic state to imply all states are aperiodic.

BUT let's not forget ??? ***(dangling nodes)***



Background/teleportation transition matrix

$$0.15 \begin{bmatrix} 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{bmatrix} + 0.85 \begin{bmatrix} 0 & 1/3 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 1/2 \\ 1/3 & 1/3 & 0 & 0 \end{bmatrix}$$

Graph transition matrix

unbiased teleportation

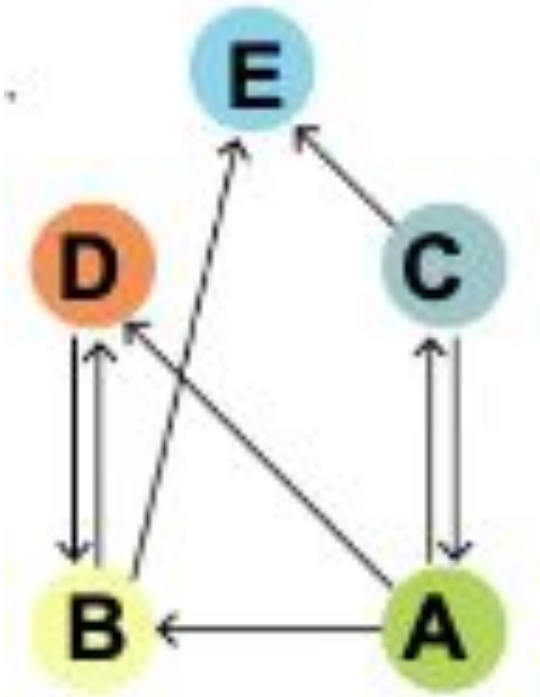
dangling node mass

mass from neighbors

$$P = \alpha \times \frac{1}{|G|} + (1-\alpha)\left(\frac{m}{|G|} + P'\right) \text{ where } \alpha = 0.15$$

Graph: with initial pagerank value of 0.2 for each page

- Calculate the PageRank of node A after one iteration of pagerank



$$P = \alpha \times \frac{1}{|G|} + (1-\alpha)\left(\frac{m}{|G|} + P'\right) \text{ where } \alpha = 0.15$$

Full Pagerank: teleportation + dangling node treatment

Part 1

Map: distribute PageRank "credit" to link targets, and pass the graph structure

{outlinks}

A	B	C	D	A
0.2	0.2/3	0.2/3	0.2/3	-
{B,C,D}	{}	{}	{}	{B,C,D}

B	D	E	B
0.2	0.2/2	0.2/2	-
{D,E}	{}	{}	{D,E}

C	A	E	C
0.2	0.2/2	0.2/2	-
{A,E}	{}	{}	{A,E}

D	B	D
0.2	0.2/1	-
{B}	{}	{B}

E	E
0.2	-
{}	{}

Reduce: gather up PageRank "credit" from multiple sources to compute new PageRank value

A	B	C	D	E
0.149	0.291	0.121	0.205	0.234
{B,C,D}	{D,E}	{A,E}	{B}	{}

Iterate until convergence

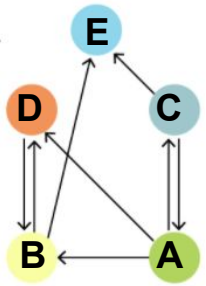
0.149 = 0.15*(1/5) + (1-0.15)*(0.2/5 + 0.2/2)

Sanity check -> 0.149 + 0.291 + 0.121 + 0.205 + 0.234 = 1.0

$$P = \alpha \times \frac{1}{|G|} + (1-\alpha)\left(\frac{m}{|G|} + P'\right) \text{ where } \alpha = 0.15$$

Part 2

Dangling Node!! Its mass has nowhere to go.



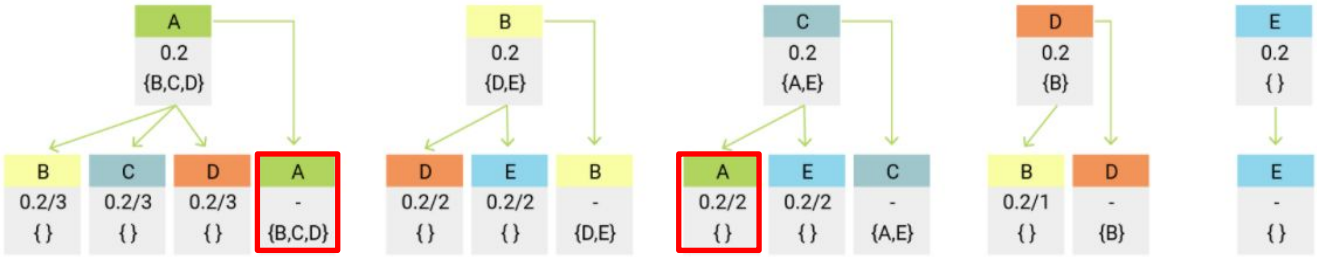
distribute uniformly: 0.2/5 where 5 is the number of nodes

Dangling Mass 0.2

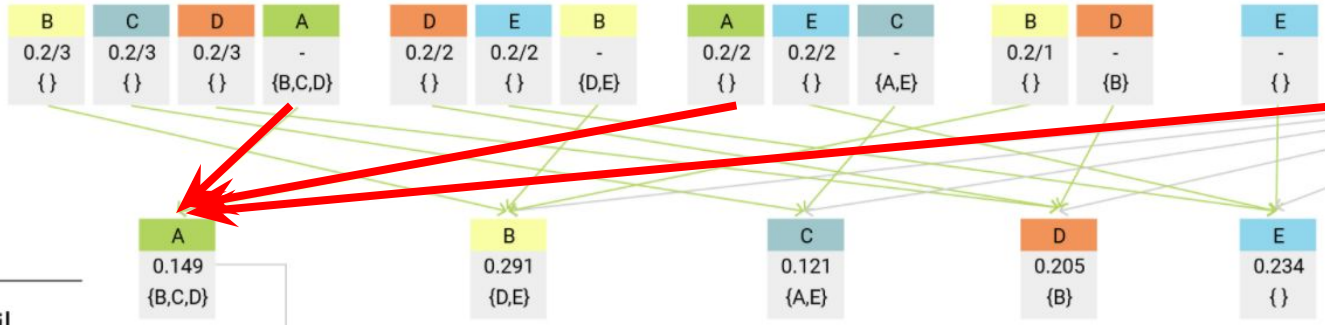
Full Pagerank: teleportation + dangling node treatment

Part 1

Map: distribute PageRank "credit" to link targets, and pass the graph structure



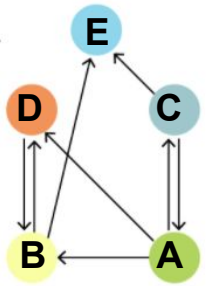
Reduce: gather up PageRank "credit" from multiple sources to compute new PageRank value



Iterate until convergence

Part 2

Dangling Node!! Its mass has nowhere to go.



distribute uniformly: 0.2/5 where 5 is the number of nodes

Dangling Mass 0.2

0.149 = 0.15*(1/5) + (1-0.15)*(0.2/5 + 0.2/2)

Sanity check -> 0.149 + 0.291 + 0.121 + 0.205 + 0.234 = 1.0

$$P = \alpha \times \frac{1}{|G|} + (1-\alpha)\left(\frac{m}{|G|} + P'\right) \text{ where } \alpha = 0.15$$

Live Session Outline

- **Page Rank**
  - Adjacency Matrices and Markov Processes
  - Calculating Steady State and The Power Method
  - Distributed PageRank
  - Dangling Nodes
  - Topic Specific PageRank
- **Bonus [Optional] Topics**
  - Contextual advertising [Optional]
  - Text as graph: TextRank [Optional]
    - Keyword extraction (from text/target pages)
    - Text Summarization

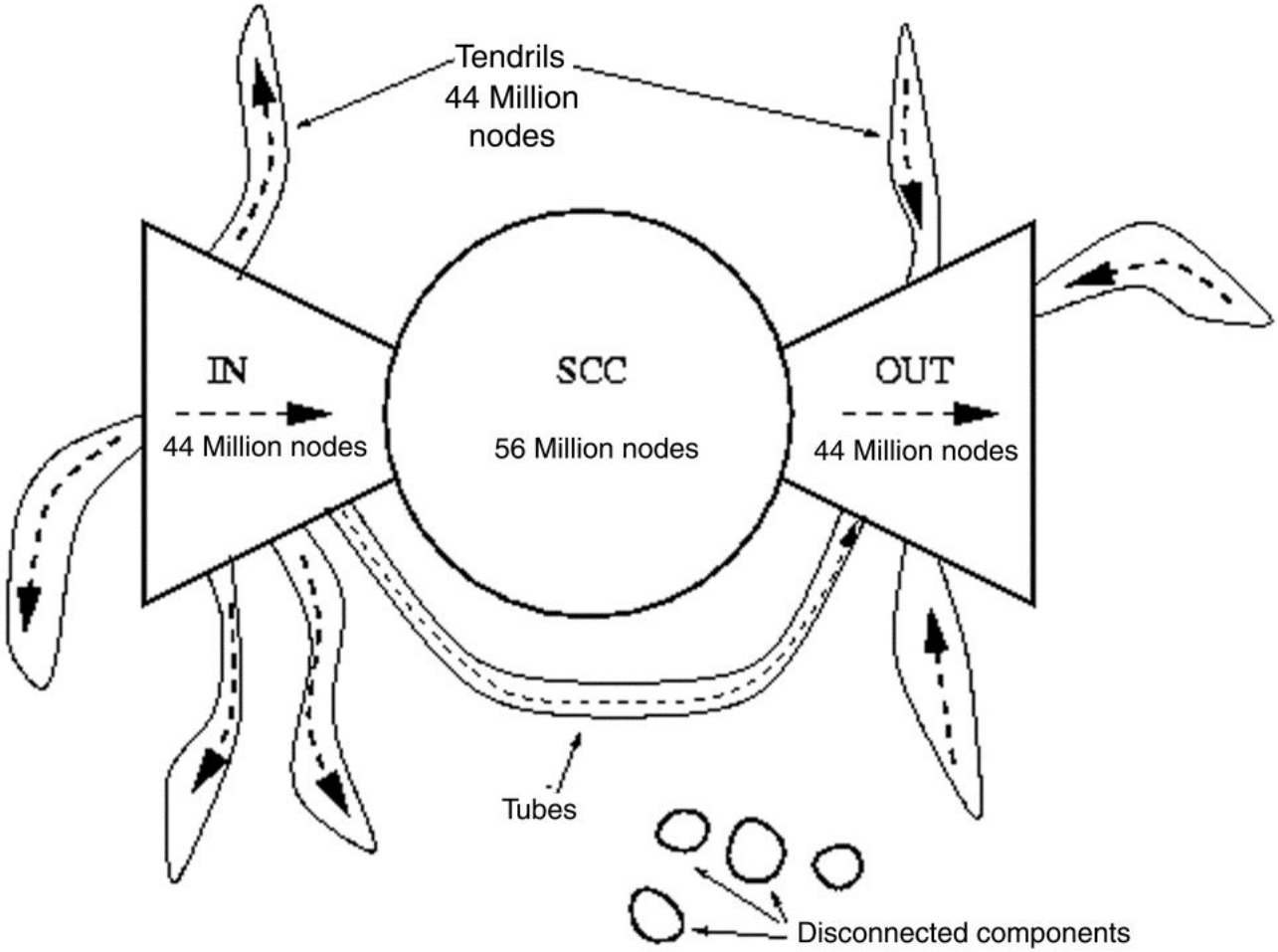


Fig. 9. Connectivity of the Web: one can pass from any node of IN through SCC to any node of OUT. Hanging off IN and OUT are TENDRILS containing nodes that are reachable from portions of IN, or that can reach portions of OUT, without passage through SCC. It is possible for a TENDRIL hanging off from IN to be hooked into a TENDRIL leading into OUT, forming a TUBE: i.e., a passage from a portion of IN to a portion of OUT without touching SCC.

http://snap.stanford.edu/class/cs224w-readings/broder00bowtie.pdf

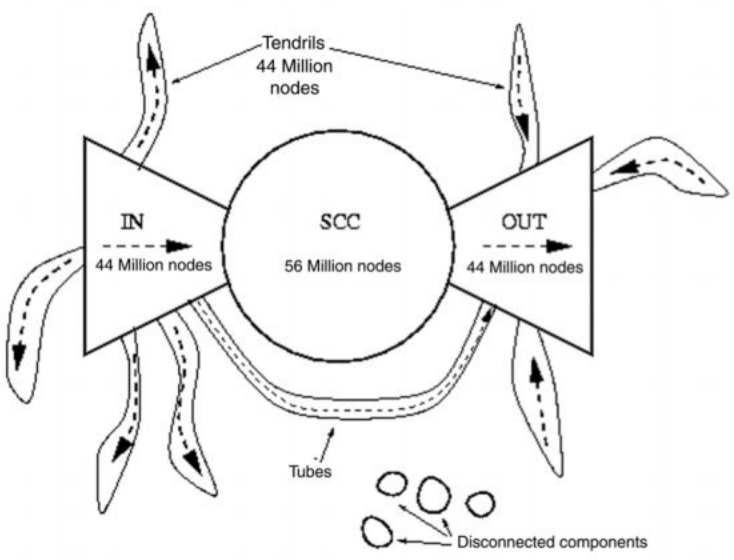


Fig. 1. The shape of the Web Graph (according to [2]) is mainly a Bowtie looking graph with a few extra parts. We will refer to the strongly connected component (SCC) as the CORE, since it more appropriately reflects on its role.

http://cs.wellesley.edu/~pmetaxas/Why_Is_the_Shape_of_the_Web_a_Bowtie.pdf

Computer Networks 33 (2000) 309–320

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

Graph structure in the Web

Andrei Broder [a], Ravi Kumar [b,*], Farzin Maghoul [a], Prabhakar Raghavan [b], Sridhar Rajagopalan [b], Raymie Stata [c], Andrew Tomkins [b], Janet Wiener [c]

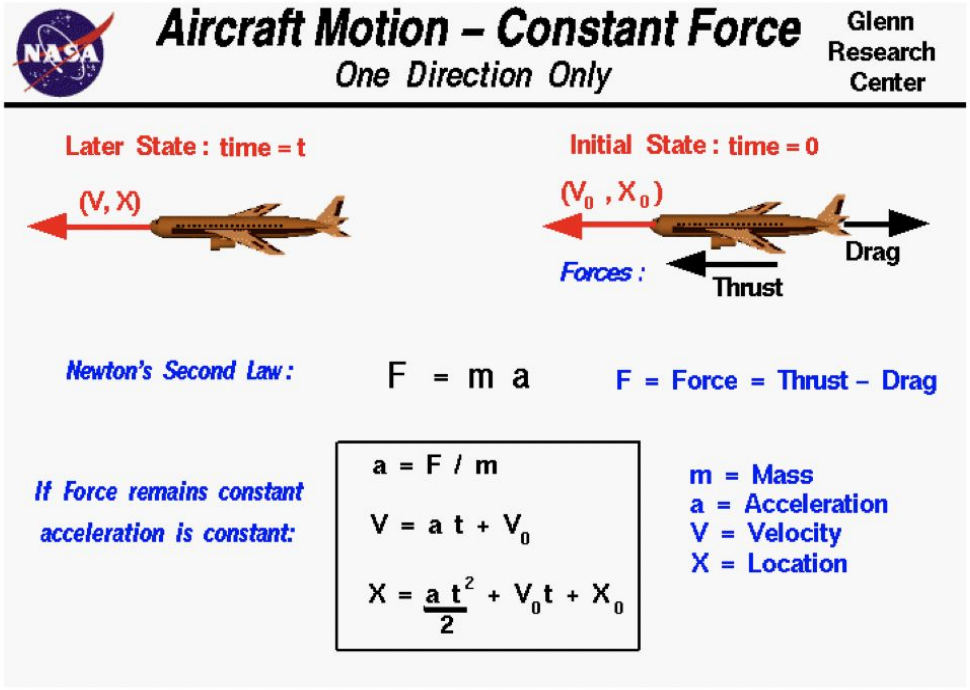
[a] *AltaVista Company, San Mateo, CA, USA*
[b] *IBM Almaden Research Center, San Jose, CA, USA*
[c] *Compaq Systems Research Center, Palo Alto, CA, USA*

Abstract

The study of the Web as a graph is not only fascinating in its own right, but also yields valuable insight into Web algorithms for crawling, searching and community discovery, and the sociological phenomena which characterize its evolution. We report on experiments on local and global properties of the Web graph using two AltaVista crawls each with over 200 million pages and 1.5 billion links. Our study indicates that the macroscopic structure of the Web is considerably more intricate than suggested by earlier experiments on a smaller scale.

Keywords: Graph structure; Diameter; Web measurement

Newton's second law: later state calculation deterministically



- Newton's second law gives us a deterministic way way to represent the dynamics of a system
  - $X = 0.5 * a * t^2 + V_o * t + X_o$
  - Deterministic system
- Markov Chains give us a great way of approximating systems which are noisy
  - $S_{t+1} = f(S_t, \text{noise})$
  - Use a Markov model
  - Tell me about future states if the system
  - Tell what may happen a few time steps into the future regardless of the starting state

https://www.grc.nasa.gov/WWW/K-12/airplane/motion.html

Velocity, Location (V, X)

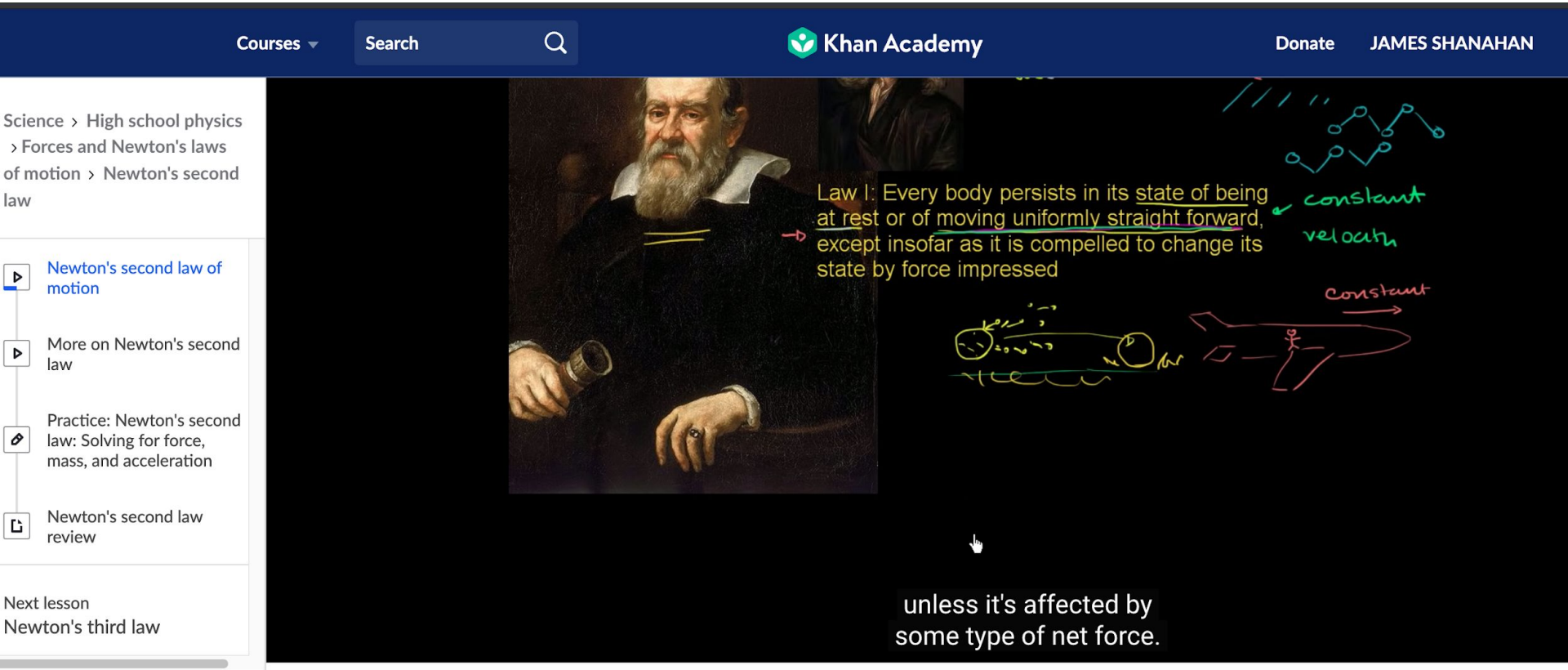
Classic mechanics versus quantum mechanics

The Newton's second law of motion states that acceleration of an object is proportional to the net force F acting on it and inversely proportional to its mass m. It is expressed with the following equation

- $a = F / m$,

where

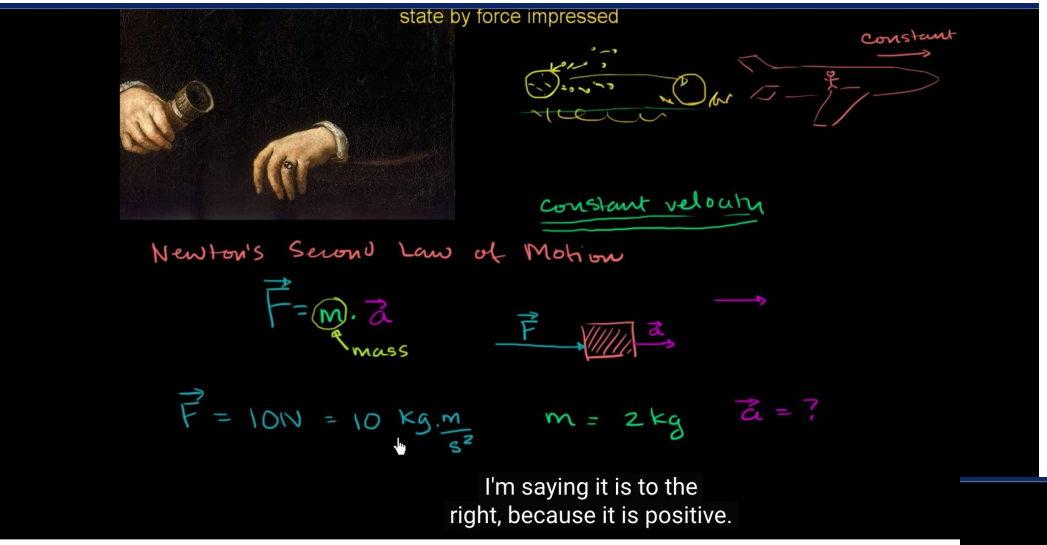
- a [m/s^2] is the acceleration of an object;
- F [N] is the force acting on an object;
- m [kg] is the mass of an object.

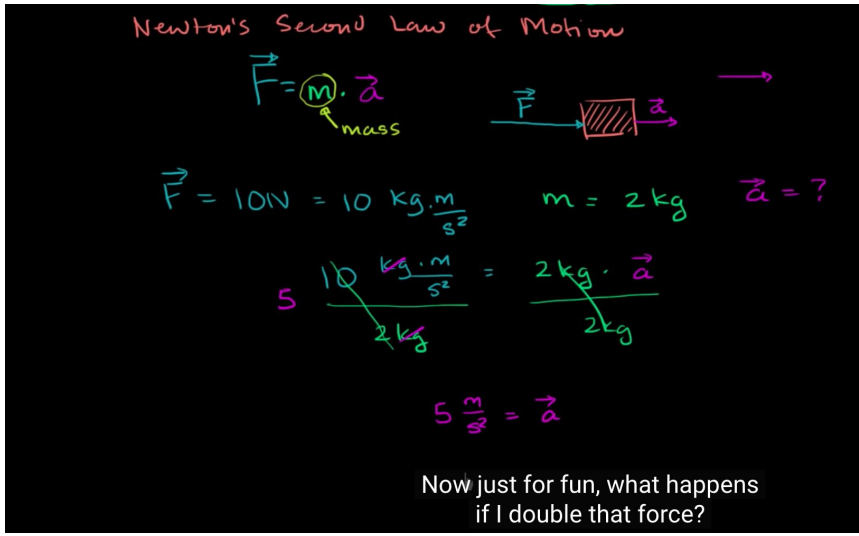


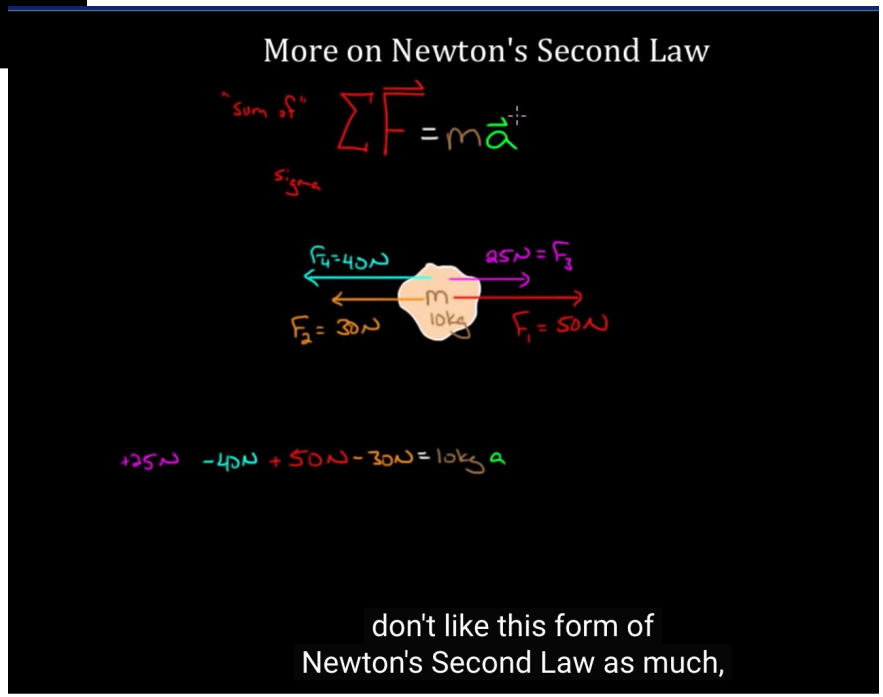
Newton's second law of motion

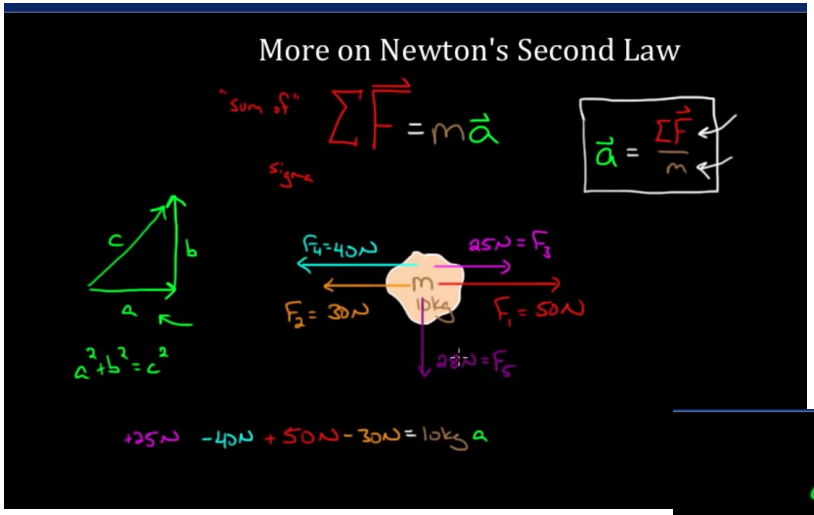
Mass is how much stuff there is…
Weight has a gravity force

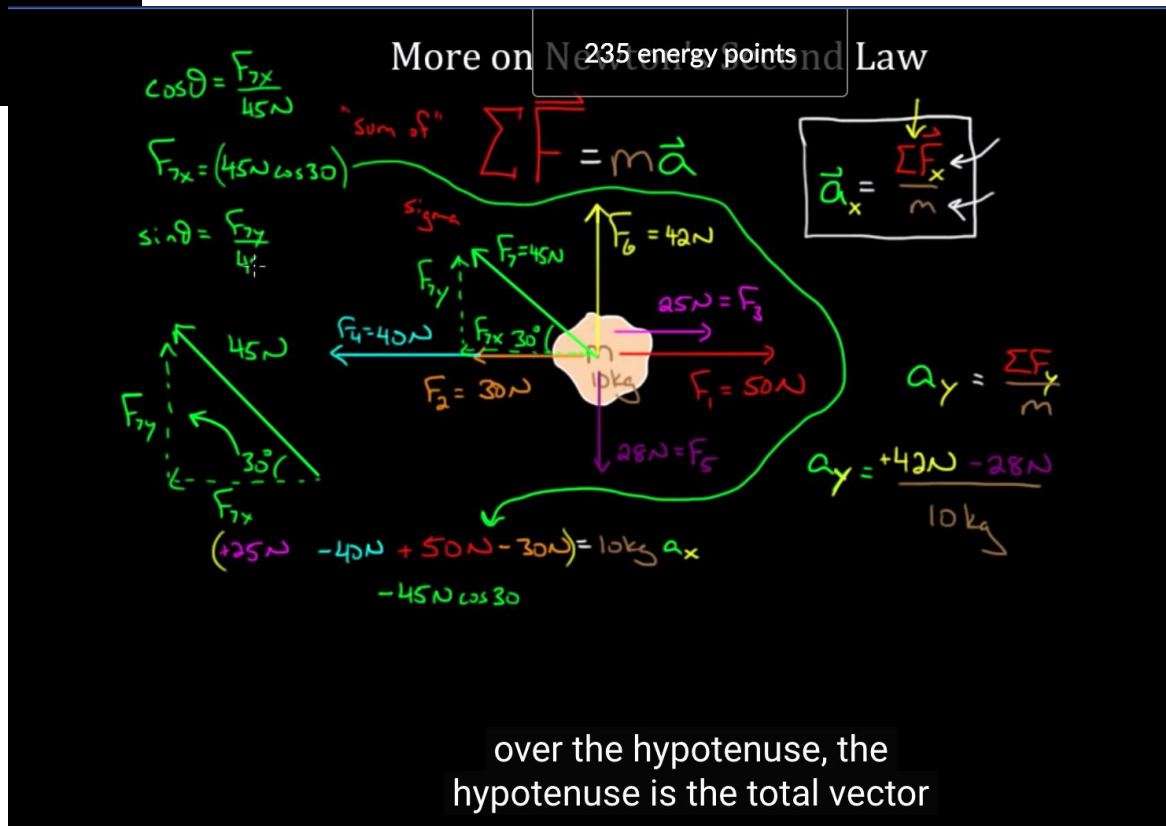
Force is to the right…so positiuve force only











235 energy points

over the hypotenuse, the hypotenuse is the total vector

https://www.khanacademy.org/science/high-school-physics/forces-and-newtons-laws-of-motion/newtons-second-law/v/more-on-newtons-second-law

Model random walk as a Markov Chain

- A Markov chain is a mathematical system that undergoes transitions from one state to another, between a finite or countable number of possible states.
  - named after Andrey Markov

- It is a random process usually characterized as memoryless: the next state depends only on the current state and not on the sequence of events that preceded it.

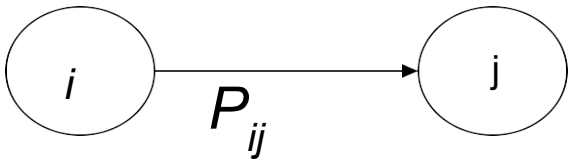
- This specific kind of "memorylessness" is called the Markov property. Markov chains have many applications as statistical models of real-world processes.

Markov chain=States+Transition Matrix

- A Markov chain consists of
  - state vector of dimension n and an
  - $n \times n$ transition probability matrix P.

- At each time step, we are in exactly one of the states.

- For $1 \leq i,j \leq n$, the matrix entry P_{ij} tells us the probability of j being the next state, given we are currently in state i.



PageRank in Linear Algebra

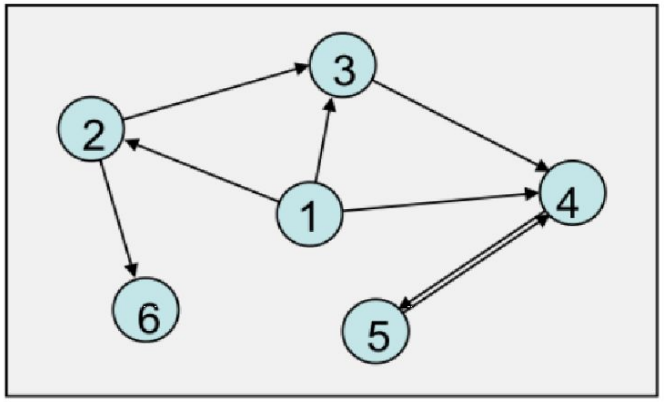


Figure 1: Example network with nodes representing websites and edges representing links

We can formalize this intuition via the process of a random walk. We would like to model a "random-surfer" who is traversing the web with uniform probability of following any link outgoing from the page the surfer is currently on. We are interested in the behavior of this random surfer in the limit as she takes an infinite number of jumps.

To express this in linear algebra, we will make a use of the adjacency matrix, A, and a out-degree matrix D, where:

$$A_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Q: transition matrix;
A: an out-degree matric

To express this in linear algebra, we will make a use of the adjacency matrix, A, and a out-degree matrix D, where:

$$A_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

$$D = \begin{pmatrix} deg(v_1) & 0 & \dots & 0 \\ 0 & deg(v_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & deg(v_n) \end{pmatrix}$$

P matrix for subsequent slides

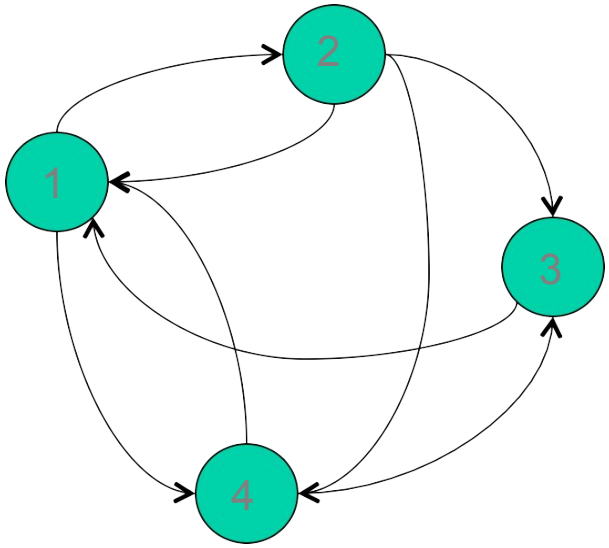
Let $Q = D^{-1}A$. This forms the transition matrix of the random-walker. Given the current state of the walker each row of the matrix gives the probability the walker will transition to each new state.

$$Q_{i,j} = \begin{cases} 1/deg(v_i) & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

It's interesting to note that Q_{ij}^k is equal to the probability of going from node i to node j in exactly k steps, in a random walk over graph G.

(k steps are the time-steps, not to be confused with hops (paths))

Math from the previous slide



$$A_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

$$D = \begin{pmatrix} deg(v_1) & 0 & \dots & 0 \\ 0 & deg(v_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & deg(v_n) \end{pmatrix}$$

a_{ij} link between node i and j exists *(out-degree of node i)*

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad D = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \quad D^{-1} = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

Transition Matrix $Q = D^{-1}A = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ 1 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix}$

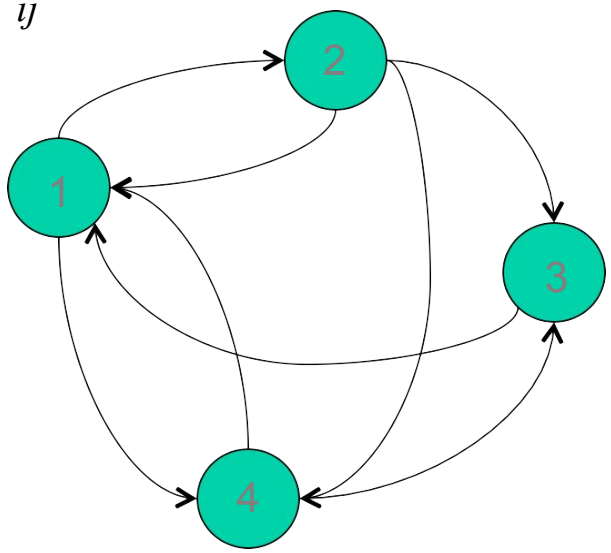
$$Q_{i,j} = \begin{cases} 1/deg(v_i) & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

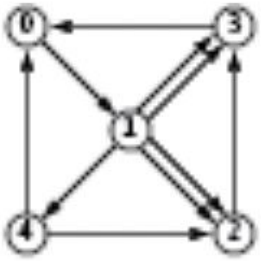
Markov chains are abstractions of random walks

- Clearly, for all each state/node i,
- An example Transition Matrix without teleportation

	1	2	3	4
1	0	0.5	0	0.5
2	0.33	0	0.33	0.33
3	1	0	0	0
4	0.5	0	0.5	0

$\sum_i P_{ij} = 1$





p

.02	.92	.02	.02	.02
.02	.02	.38	.38	.20
.02	.02	.02	.92	.02
.92	.02	.02	.02	.02
.47	.02	.47	.02	.02

probability of surfing from i to 2 in one move

probability of surfing from 1 to i in one move

ranks[] p[][] newRanks[]

first move

[1.0 0.0 0.0 0.0 0.0] * | .02 .92 .02 .02 .02 | .02 .02 .38 .38 .20 | .02 .02 .02 .92 .02 | .92 .02 .02 .02 .02 | .47 .02 .47 .02 .02 | = [.02 .92 .02 .02 .02]

probabilities of surfing from 0 to i in one move

second move

probabilities of surfing from i to 2 in one move

probabilities of surfing from 0 to i in one move

[.02 .92 .02 .02 .02] * | .02 .92 .02 .02 .02 | .02 .02 .38 .38 .20 | .02 .02 .02 .92 .02 | .92 .02 .02 .02 .02 | .47 .02 .47 .02 .02 | = [.05 .04 .36 .37 .19]

probability of surfing from 0 to 2 in two moves (dot product)

probabilities of surfing from 0 to i in two moves

third move

probabilities of surfing from 0 to i in two moves

[.05 .04 .36 .37 .19] * | .02 .92 .02 .02 .02 | .02 .02 .38 .38 .20 | .02 .02 .02 .92 .02 | .92 .02 .02 .02 .02 | .47 .02 .47 .02 .02 | = [.44 .06 .12 .36 .03]

probabilities of surfing from 0 to i in three moves

. . .

20th move

probabilities of surfing from 0 to i in 19 moves

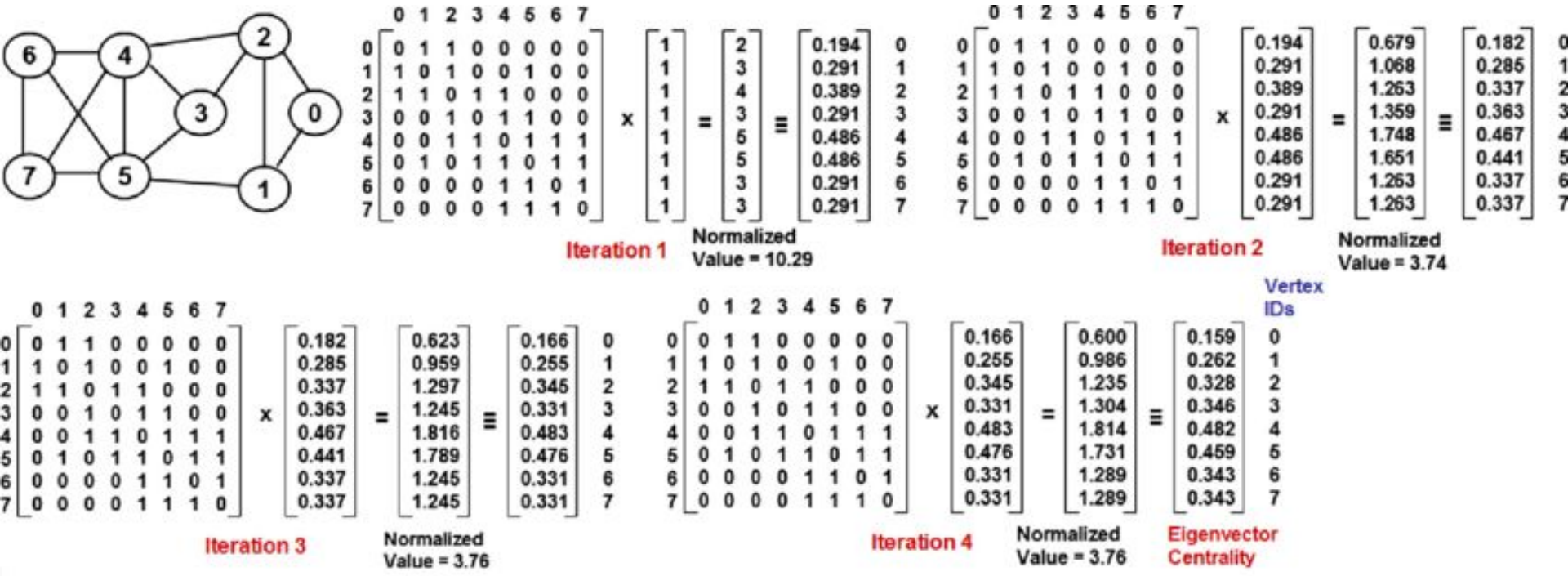
[.27 .26 .15 .25 .07] * | .02 .92 .02 .02 .02 | .02 .02 .38 .38 .20 | .02 .02 .02 .92 .02 | .92 .02 .02 .02 .02 | .47 .02 .47 .02 .02 | = [.27 .26 .15 .25 .07]

probabilities of surfing from 0 to i in 20 moves (steady state)

The power method for computing page ranks (limit values of transition probabilities)

https://introcs.cs.princeton.edu/python/16pagera...

Power iteration method for finding the top eigenvector



Transition matrix; stationary vector

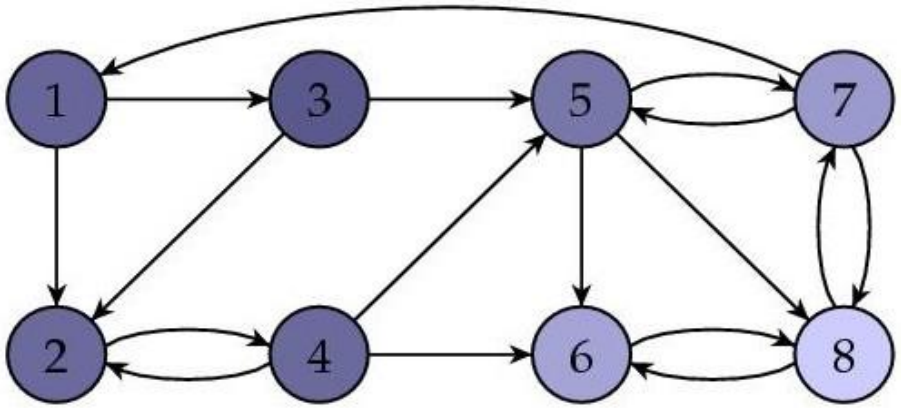
Graph in matrix form

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 0 \\ 1/2 & 0 & 1/2 & 1/3 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 1 & 1/3 & 0 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix}$$

with stationary vector

$$I = \begin{bmatrix} 0.0600 \\ 0.0675 \\ 0.0300 \\ 0.0675 \\ 0.0975 \\ 0.2025 \\ 0.1800 \\ 0.2950 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix}$$

This shows that page 8 wins the popularity contest. Here is the same figure with the web pages shaded in such a way that the pages with hig PageRanks are lighter.



Lighter color means higher PR

Computing PR Via Power Method

The method is founded on the following general principle that we will soon investigate.

> ***General principle:*** *The sequence I^k will converge to the stationary vector I.*

We will illustrate with the example above.

$I H = I^1$

$I^1 H = I^2$

....

I^0	I^1	I^2	I^3	I^4	...	I^{60}	I^{61}
1	0	0	0	0.0278	...	0.06	0.06
0	0.5	0.25	0.1667	0.0833	...	0.0675	0.0675
0	0.5	0	0	0	...	0.03	0.03
0	0	0.5	0.25	0.1667	...	0.0675	0.0675
0	0	0.25	0.1667	0.1111	...	0.0975	0.0975
0	0	0	0.25	0.1806	...	0.2025	0.2025
0	0	0	0.0833	0.0972	...	0.18	0.18
0	0	0	0.0833	0.3333	...	0.295	0.295

first eigenvector

It is natural to ask what these numbers mean. Of course, there can be no absolute measure of a page's importance, only relative measures for comparing the importance of two pages through statements such as "Page A is twice as important as Page B." For this reason, we may multiply all the importance rankings by some fixed quantity without affecting the information they tell us. In this way, we will always assume, for reasons to be explained shortly, that the sum of all the popularities is one.

6.1.1. The Power Iteration method produces the steady state vector, which should match the PageRank result above. Or does it? ¶

```
# Power Iteration helper function - RUN THIS CELL AS IS
import numpy as np
def power_iteration(xInit, tMatrix, nIter, verbose = True):
    """ Calculate one eigen vector"""
    state_vector = xInit

    for ix in range(nIter):
        new_state_vector = state_vector@tMatrix
        state_vector = new_state_vector

        if verbose:
            print(f'Step {ix}: {state_vector}')

    return state_vector

def deflation(tMatrix, eigen_vector):
    #updated_tMatrix = tMatrix - eigen_vector
    updated_tMatrix = None
    return updated_tMatrix

def pca(tMatrix, num_of_eigenvectors_reqd):
    for i in range(num_of_eigenvectors_reqd):
        xInit = np.array([1,0,0,0])
        eigen_vector = power_iteration(xInit, tMatrix, nIter, verbose = True)
        tMatrix = deflation(tMatrix, eigen_vector)

xInit = np.array([1,0,0,0])
```

Example:

Adjacency Matrix → Transition Matrix

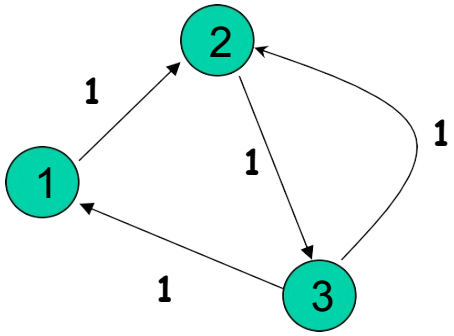
0	1	0
0	0	1
1	1	0

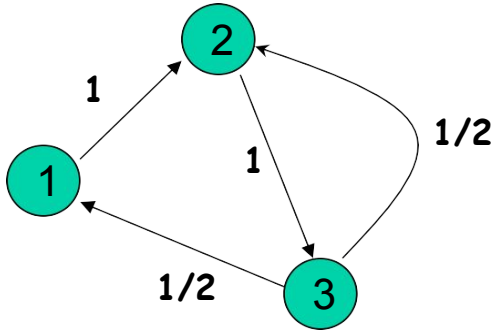
Adjacency matrix A

0	1	0
0	0	1
1/2	1/2	0

Transition matrix P

Right row stochastic





A right stochastic matrix is a real, positive, square matrix, with each row summing to 1.

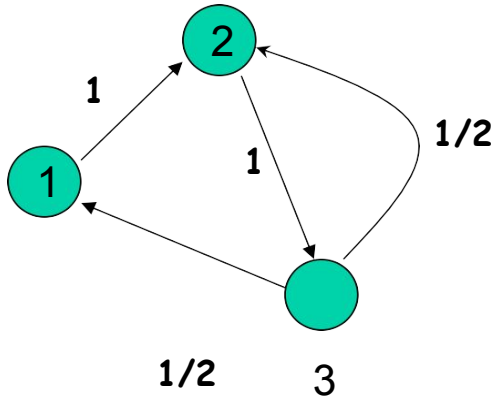
What goes in and what goes out

For <u>right</u> stochastic matrices:

- **Inbound → on the side**
- **Outbound → across the top**
- **In Multiplication**
  - **States x^{before} are on the left**
  - **P is on the <u>right</u>**
    - $x^{before}P = x^{after}$

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1/2 & 1/2 & 0 \end{bmatrix}$$

Weights contributing to x_1^{before}



Connecting the dots: PCA – Power Iterations - SVD

- https://www.cs.unc.edu/~coombe/research/phd/svd.pdf
- **Computer eigenvectors**
  - SVD (in lecture 14 for recommender systems)
  - Power iteration described below
- **Repeat**
  - Compute the eigenvector using Power Iteration
  - To compute the subsequent eigenvalues, the current eigenvector is removed from A in a process called Deflation.
- **Until you have enough of eigenvectors**

PAGERANK: from random walks to pagerank

- **View pages as states, and webGraph as a transition matrix**
- **A Markov process whose steady state distribution tells us about which pages we "spend a lot of time on".**
- **Making some minor adjustments to the transition matrix**
  - **stochasticity adjustment:** adjustment to deal with dangling nodes (sinks)); In this adjustment, the 0 rows of matrix H are replaced with $1/n$ making H stochastic. This adjustment now allows the random surfer to hyperlink to any page at random after entering a dangling node.
  - **Primitivity:** adjustment via teleportation
- **We can then use the power iteration method**
  - Top eigenvector, all positive values; eigenvalue $=1$
- **This is a proxy for popularity and is a very discriminating feature for websearch in the context of webpages**
- **Variations: Personalized pagerank,text summarization, fraud detection**

Live Session Outline

- **Page Rank**
  - Adjacency Matrices and Markov Processes
  - Calculating Steady State and The Power Method
  - Distributed PageRank
  - Dangling Nodes
  - Topic Specific PageRank
- **Bonus [Optional] Topics**
  - Contextual advertising [Optional]
  - Text as graph: TextRank [Optional]
    - Keyword extraction (from text/target pages)
    - Text Summarization

Pagerank:"measures" relative importance

- Connect all the dots: Random surfer → markov process

- Adapt the machinery of Markov processes to give us a principled approach to calculate the pagerank of each webpage; it nothing more than the steady state probability distribution of the markov process underlying the random surfer model of web navigation

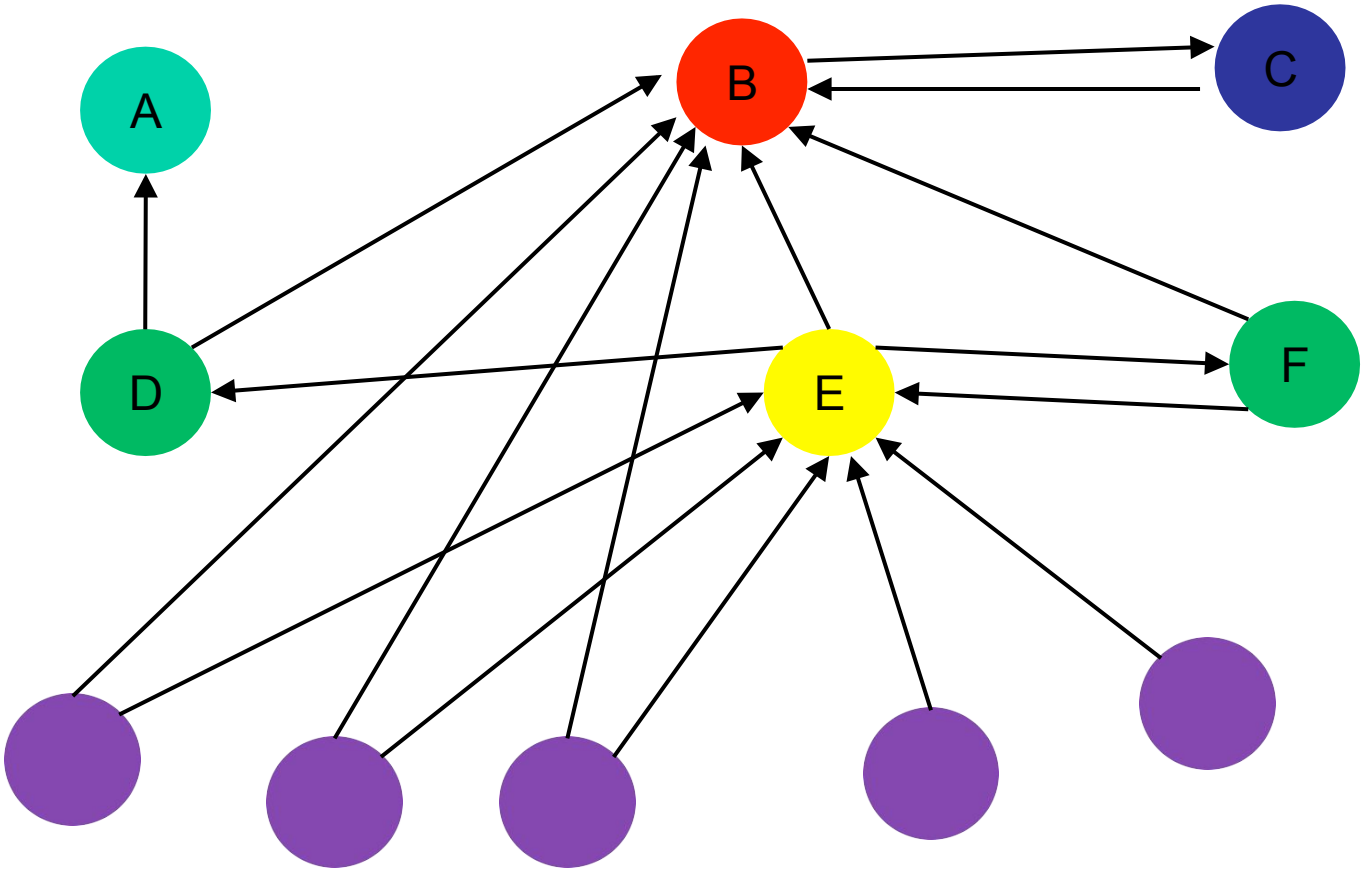
- ***PageRank is a link analysis algorithm that "measures" relative importance of each within the webgraph.***

$$v_0 P = v_1$$

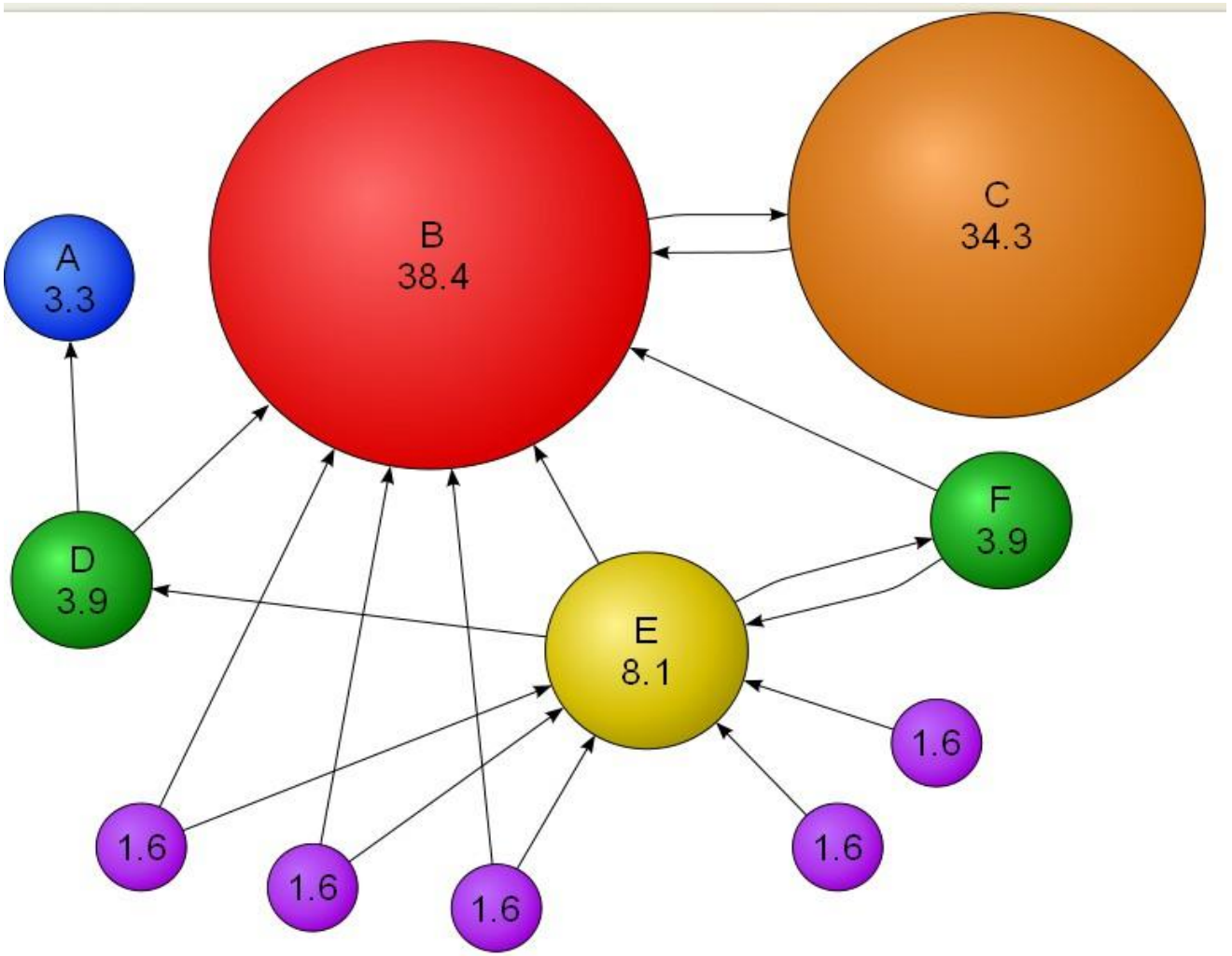
P is Right row stochastic

PageRank example

15% probability of a random jump + dangling nodes



PageRank: Steady State Pr of being at a Page



38.4
+34.3
+3.3
+3.9
+8.1
+3.9
+1.6
+1.6
+1.6
+1.6
+1.6
=

100

Transition matrix; stationary vector

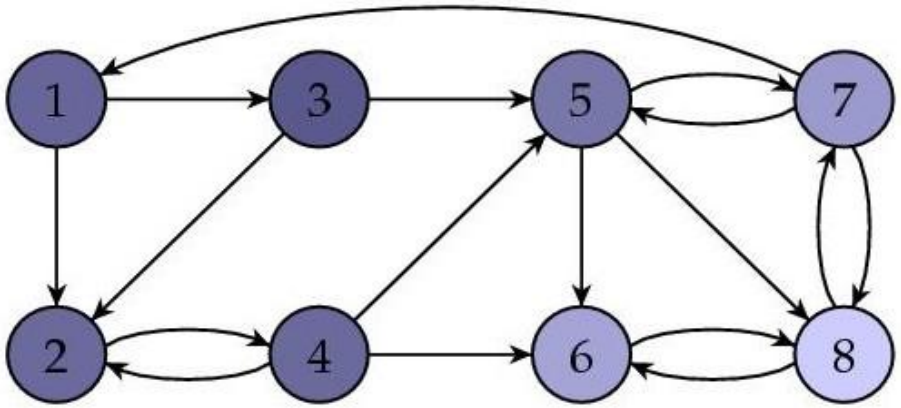
Graph in matrix form

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 0 \\ 1/2 & 0 & 1/2 & 1/3 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 1 & 1/3 & 0 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix}$$

with stationary vector

$$I = \begin{bmatrix} 0.0600 \\ 0.0675 \\ 0.0300 \\ 0.0675 \\ 0.0975 \\ 0.2025 \\ 0.1800 \\ 0.2950 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix}$$

This shows that page 8 wins the popularity contest. Here is the same figure with the web pages shaded in such a way that the pages with hig PageRanks are lighter.



Lighter color means higher PR

Computing PR Via Power Method

The method is founded on the following general principle that we will soon investigate.

> ***General principle:*** *The sequence I^k will converge to the stationary vector I.*

We will illustrate with the example above.

$I H = I^1$

$I^1 H = I^2$

....

I^0	I^1	I^2	I^3	I^4	...	I^{60}	I^{61}
1	0	0	0	0.0278	...	0.06	0.06
0	0.5	0.25	0.1667	0.0833	...	0.0675	0.0675
0	0.5	0	0	0	...	0.03	0.03
0	0	0.5	0.25	0.1667	...	0.0675	0.0675
0	0	0.25	0.1667	0.1111	...	0.0975	0.0975
0	0	0	0.25	0.1806	...	0.2025	0.2025
0	0	0	0.0833	0.0972	...	0.18	0.18
0	0	0	0.0833	0.3333	...	0.295	0.295

It is natural to ask what these numbers mean. Of course, there can be no absolute measure of a page's importance, only relative measures for comparing the importance of two pages through statements such as "Page A is twice as important as Page B." For this reason, we may multiply all the importance rankings by some fixed quantity without affecting the information they tell us. In this way, we will always assume, for reasons to be explained shortly, that the sum of all the popularities is one.

Computing PR Via Power Method

The method is founded on the following general principle that we will soon investigate.

General principle: The sequence I^k will converge to the stationary vector I.

We will illustrate with the example above.

I H =

I^1H =

....

It is natural to ask
comparing the impor
all the importance ra
to be explained shor

e measures for
e may multiply
ne, for reasons

Why does this work?

Will it always work?

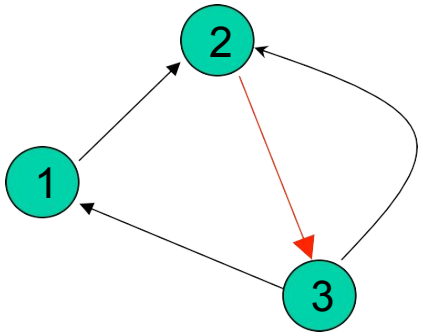
Lets look at the necessary restrictions on the Graph...

Interlude:

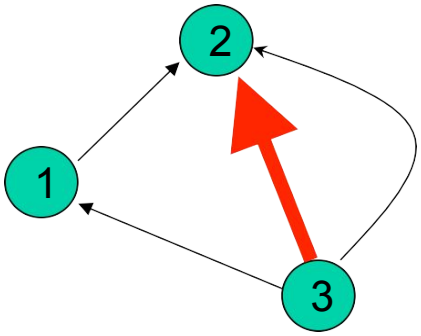
Well behaved graphs - Irreducible

- **Irreducible: There is a path from every node to every other node.**

Node 2 is a Dangling node



Irreducible



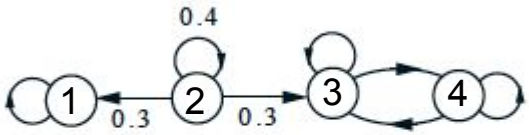
Not irreducible

Can not reach nodes 1 and 3 from node 2. (aka Dangling node, i.e., a node with no out links)

Irreducibility

A Markov chain is said to be **irreducible** if its state space is a single communicating class; in other words, if it is possible to get to any state from any state.

- Does the limit depend on initial state?



$r_{1\,1}(n) =$ 1, for all n

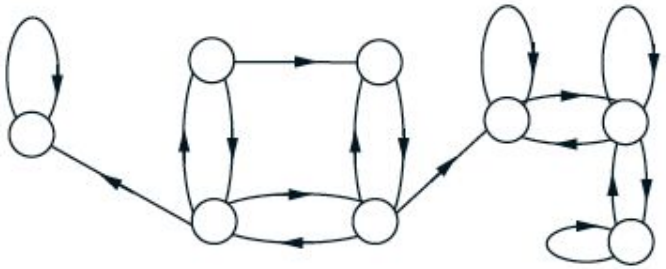
$r_{3\,1}(n) =$ 0, for all n

$r_{2\,1}(n) =$ ½ as n approaches infinity

The probability of being in a particular state is very much affected by where you started from.

Recurrent and transient states

- State i is **recurrent** if:
  starting from i,
  and from wherever you can go,
  there is a way of returning to i
- If not recurrent, called **transient**



– i transient:
  $P(X_n = i) \to 0$,
  i visited finite number of times

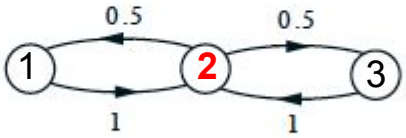
- **Recurrent class**:
  collection of recurrent states that
  "communicate" with each other
  and with no other state

dangling nodes make for transient states.

Periodicity

Generic convergence questions:

- Does $r_{ij}(n)$ converge to something?



n odd: $r_{22}(n) = 0$ n even: $r_{22}(n) = 1$

where n *is the number of transitions*

If you start at **2**, you have a 50/50 chance of going left or right. So there is some randomness. But this randomness is limited. No matter whether you go left or right, you always come back to **2** in the next step. You go out, you go in, you go out, you go in - there is a periodic pattern that gets repeated.

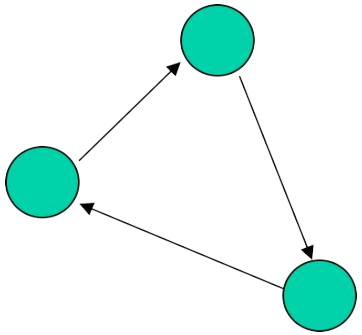
It means that if you start at state **2**, after an even number of steps, you are certain to be back at state **2**, so the probability is 1. If the number of transitions is odd, there is no way you can be at state **2**. At odd transitions, you will be at either the left or right, so the probability of being at state **2** is 0.

As n (number of transitions) goes to infinity, this state **2** probability **does not converge** to anything. It keeps alternating between 0 and 1.

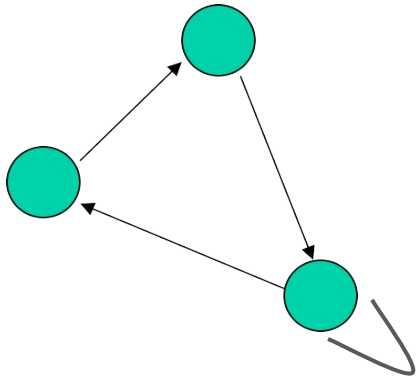
Interlude:

Well behaved graphs: Must be Aperiodic

- **Aperiodic: The GCD (greatest common divisor) of all cycle lengths is *1*. The GCD is also called period.**



Periodicity = 3
(Every state returns to its starting point after *3* steps)



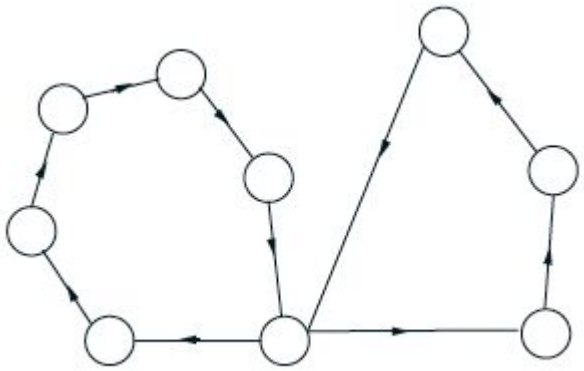
Aperiodic: The self-loops ($G_{ii} > 0$ for all i) create aperiodicity.

A Markov chain is aperiodic if every state is aperiodic.
An irreducible Markov chain only needs one aperiodic state to imply all states are aperiodic.

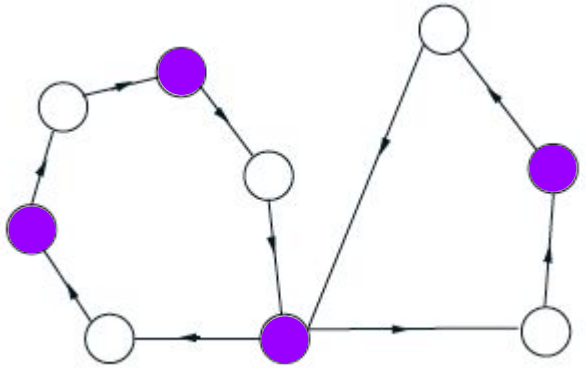
http://en.wikipedia.org/wiki/Aperiodic_graph

FUN EXAMPLE

Is this structure aperiodic?

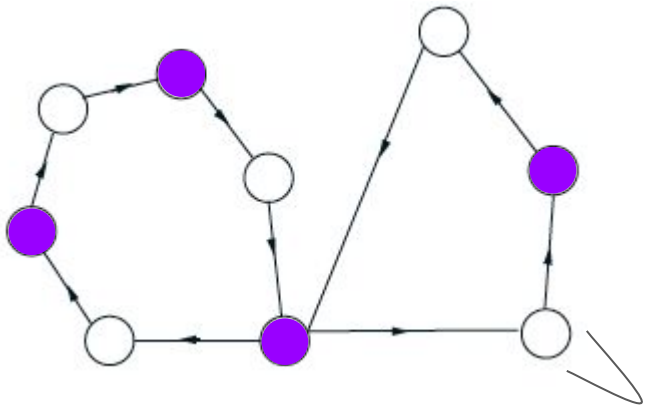


NO!
If you are in a purple state, you can only go to a white state, and vs. Again, as N (number of transitions) goes to infinity, the probability of being in the purple or white state **does not converge** to anything. It keeps alternating between 0 and 1.



If you are at a purple state, then the probability of going into a white state is 1. But if you are in a white state, then the probability of going to a white state is 0. No matter how many times you walk this graph (ie, ad infinitum), these probabilities will continue to oscillate, and will never settle on a steady state probability. Here the initial condition has an influence on the probabilities of being in each state.

A single self loop fixes this. You can now got to any color state in both even and odd times, and the probabilities can converge.



Visit frequency interpretation

Looking at it another way, we can think of the probabilities of being in a certain state, in terms of frequencies. The more more frequently you visit that state, the higher the probability of being in that state.

The steady state convergence theorem

Think of two copies of the chain that start at different initial states.

The state moves randomly, and as the state moves around randomly starting from the two initial states, on a random trajectory… as long as you have a single recurrent class, and you don't have periodicity, at some point those two states (those trajectories) are going to collide. Just because there's enough randomness there. After the state becomes the same, the future of those trajectories, probabilistically, is the same, because they both started in the same state.

So this means that the initial conditions stopped having any influence.

Perron Frobenius Theorem

- If a Markov chain is irreducible and aperiodic then the largest eigenvalue of the stochastic transition matrix will be equal to *1* and all the other eigenvalues will be strictly less than *1*.

- The **Perron Frobenius Theorem**
  - Let the eigenvalues of P be $\{\sigma_i | \ i=0:n-1\}$ in non-increasing order of σ_i
  - $\sigma_0 = 1 > |\sigma_1| \geq |\sigma_2| \geq \ldots \geq |\sigma_n|$

- The eigenvector v_0 corresponding to eigen value σ_0, is invariant (steady) under the the markov process:

$$v_0 P = v_0$$

- v_0 is the steady state probability

The Iterative Power Method

- If a markov chain is irreducible and aperiodic then the steady state probability vector

$$v_0 P = v_0$$

- Can be found by the **Iterative Power Method**, which tells us that for any probability state $\boldsymbol{x}$:

$$lim_{n \to \infty} \, xP^n = v_0$$

Proof: Iterative Power Method

- Assuming the matrix P is diagonalizable* we can write the vector in terms of the eigenbasis $\{v_i\}$:

$$x = \sum_{i=0,\ldots,n} x_i v_i = x_0 v_0 + \sum_{i=1,\ldots,n} x_i v_i$$

(x_i are the weights of vector x in the eigenbasis v_i)

- So multiplying by the Markov matrix k times:

$$xP^k = x_0 v_0 P^k + \sum_{i=0,\ldots,n} x_i v_i P^k$$
$$= x_0 v_0 \sigma_0^{\,k} + \sum_{i=0,\ldots,n} x_i v_i \sigma_i^{\,k}$$

These are all less than *1* so → *0* for large *k*

- So for for large enough k:

$$xP^k \rightarrow v_0$$

This is = *1*

- (Since ***P*** is a right stochastic matrix, the vector length is preserved under P so, $x_0 = 1$.)

*Even P is not diagonalizable, we can complete the proof more generally with Peron-projections.

PCA/SVD with Power Iterations

- https://www.cs.unc.edu/~coombe/research/phd/svd.pdf

- **Computer eigenvectors**
  - SVD (in lecture 14 for recommender systems)
  - Power iteration described below

- **Repeat**
  - Compute the eigenvector using Power Iteration
  - To compute the subsequent eigenvalues, the current eigenvector is removed from A in a process called Deflation.

- **Until you have enough of eigenvectors**

Implications of the Perron Frobenius Theorem

- So if a markov chain is irreducible and aperiodic then then there exists a vector that is invariant (steady state distribution)

- If the Markov chain is time-homogeneous, then the transition matrix P is the same after each step, so the k-step transition probability can be computed as the k^{th} power of the transition matrix, P^k.

- These results imply that for a well behaved graph there exists an unique stationary distribution.
- If we have a primitive transition matrix, then we can use power method.

PageRank Computation

- **Target**
  - Solve the steady-state probability vector v_0, which is the PageRank of the corresponding Web page.
  - $v_0 P = \lambda v_0$, λ is *1* for stochastic matrix.
- Method
  - Power iteration.
  - Given an initial probability distribution vector x
  - $x^0 P = x^1$, $x^1 P = x^2$ … Until the probability distribution x^k converges. (Variation in the computed values are below some predetermined threshold.)

So what we need is ***Matrix multiplication at scale***

PageRank

This means we now will use the following matrix to parameterize our Markov chain describing our random walker.

$$P = \alpha\Lambda + (1-\alpha)Q$$

where $0 < \alpha < 1$ is the teleport probability, and Λ is a rank 1 matrix whose rows correspond to the distribution of where a teleporting surfer arrives.
We are looking for a row vector π of node probabilities such that:

$$\pi P = \pi$$

We normally solve problems like this via power iteration. That algorithm is very simple. We initialize $V^{(0)}$ to any initial distribution -for example to the uniform vector $V^{(0)} = [1/n, \ldots, 1/n]$. Then we follow a converging recurrence:

$$V^{(k+1)} = V^{(k)}P = V^{(0)}P^{k+1}$$

V^k is the top eigenvector, also the pagerank vector, also the steady state vector.

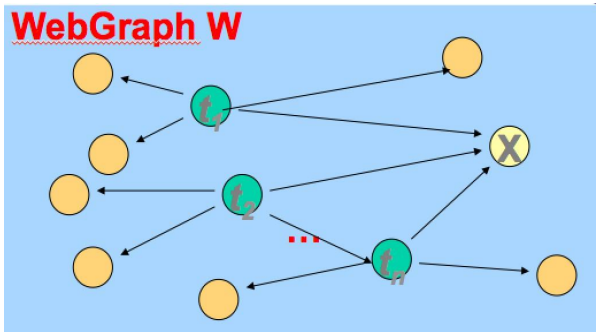
P is the transition matrix. P^{k+1} is the steady state matrix.

(The first row of P^{k+1} is V^k)

Calculating **PageRank** *PR(x)*:

Given page (node) x with links from inbound nodes $t_1 \ldots t_n$, where:

- $PR(x|W)$ means "Page Rank of x given W."
- $C(t)$ is the out-degree of t
- α is probability of random jump
- $|W|$ is the total number of nodes in the graph
- v is a biased **teleport vector** *(topic specific pagerank for example)*



We calculate the Pagerank with the following recipie:

1. Start with an arbitrary state assignment **x** for every node (i.e. $\sum_{i=0,\ldots,n} x_i = 1$)
2. $PR(x) := x_i$ by **repeatedly** updating with the following equation:

Last iteration's Page Rank Values

Updated Page Rank values

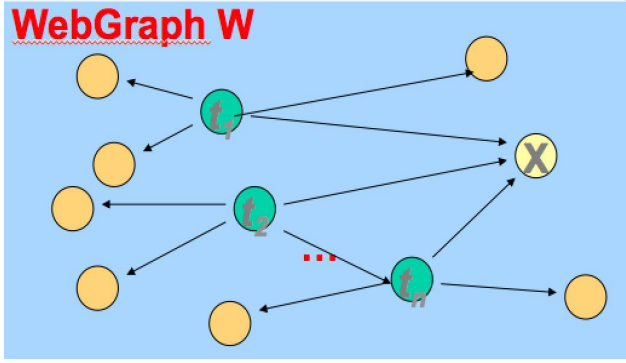
$$PR(x|W) = \alpha\left(\frac{1}{|W|}\right) + (1-\alpha)\sum_{i=1}^{n}\frac{PR(t_i)}{C(t_i)}$$

Biased PageRank, if we want to use a non-uniform teleport vector **v**

$$PR(x|W,v) = \alpha v + (1-\alpha)\sum_{i=1}^{n}\frac{PR(t_i)}{C(t_i)}$$

http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html

Iterative **PageRank** $PR(x)$:



$$PR(x|W) = \alpha\left(\frac{1}{|W|}\right) + (1-\alpha)\sum_{i=1}^{n}\frac{PR(t_i)}{C(t_i)}$$

$$PR(x|W,v) = \alpha v + (1-\alpha)\sum_{i=1}^{n}\frac{PR(t_i)}{C(t_i)}$$

Last iteration's Page Rank Values

Updated Page Rank values

Updating with (one) of these equations is equivalent to multiplying by P using a teleportation factor of α.

$$PR(x^{(k+1)}|W) = P^T x^{(k)} = x^{(k)}((1-\alpha)H + \alpha T)$$

Example:

$$P^T = 0.85 \times \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 1/3 & 1/2 & 1/5 \\ 1/2 & 0 & 0 & 1/2 & 1/5 \\ 1/2 & 1 & 0 & 0 & 1/5 \\ 0 & 0 & 1/3 & 0 & 1/5 \\ 0 & 0 & 1/3 & 0 & 1/5 \end{bmatrix} \end{array} + \alpha = .15 \quad 0.15 \times \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{bmatrix} \end{array}$$

Hyperlink Matrix

Teleporting Matrix

PageRank Algorithm: Apply Power iteration method: after k iterations $\pi = (P^T)^k\pi$
Where π is the steady state distribution vector consisting of: π_i = pagerank of page i; H is the hyperlink graph and T is the teleportation matrix.

Summary: from random walks to pagerank

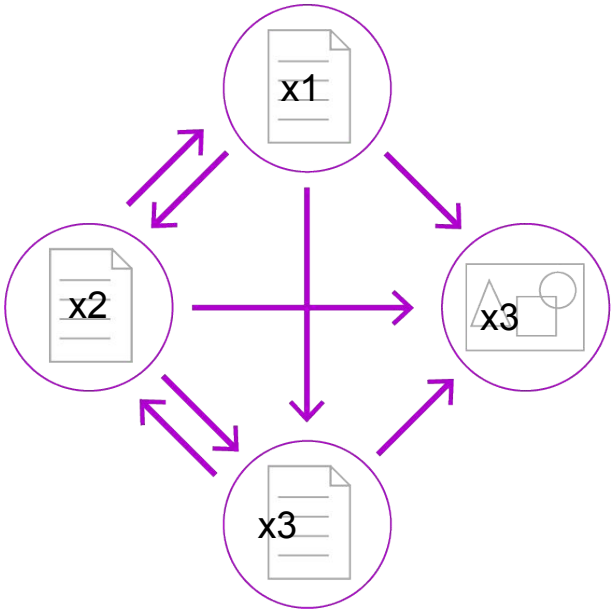
- **Simulation mode versus numerically (power iteration)**
  - View pages as states, and webGraph as a transition matrix$\rightarrow A$ Markov process who steady state distribution tells about which pages we spend a lot of time on.
- **This is a proxy for popularity and is a very discriminating feature for in search**
- **Making some minor adjustments to the transition matrix**
  - (stochasticity adjustment to deal with dangling nodes (sinks))
  - Primitivity adjustment via teleportation
- **We can then use the power iteration method**
  - (first eigenvector, all positive values; eigenvalue =*1*)
- **TAKING IT FURTHER:**
  - **Personalized pagerank**
  - **Other domains: text summarization**

Using Pagerank

- **Preprocessing:**
  - Given graph of links, build matrix P.
  - From it compute $\boldsymbol{v}$.
  - The entry a_i is a number between 0 and 1: the pagerank of page i.
- **Query processing:**
  - Retrieve pages matching the query.
  - Rank them by their pagerank.
  - Order is query-*independent*.
- **What is a good way to order Documents within an index server?**

Live Session Outline

- **Page Rank**
  - Adjacency Matrices and Markov Processes
  - Calculating Steady State and The Power Method
  - Distributed PageRank
  - Dangling Nodes
  - Topic Specific PageRank
- **Bonus [Optional] Topics**
  - Contextual advertising [Optional]
  - Text as graph: TextRank [Optional]
    - Keyword extraction (from text/target pages)
    - Text Summarization



Teleportation adjustments for PageRank

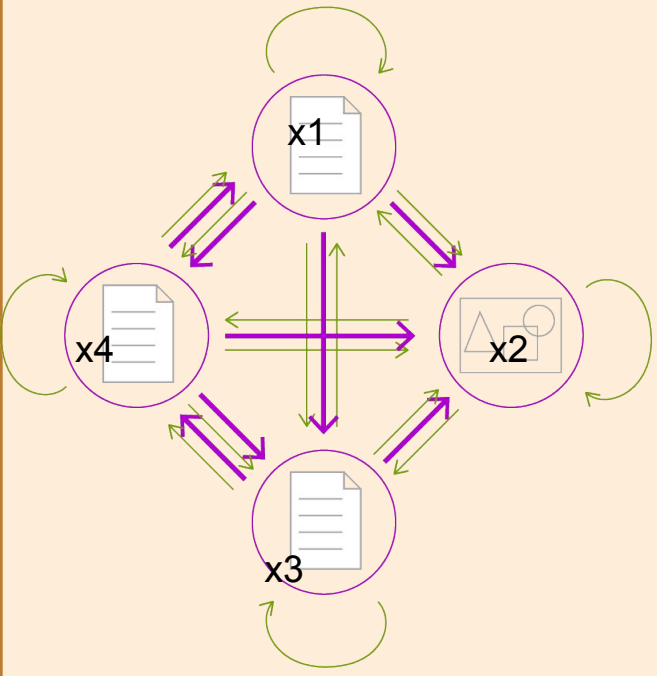
Random Jump factor adds a small amount of probability to each node to teleport to any other node.

Both issues of periodicity and irreducibility are solved at once.

1: Irreducibility/Reachable A Markov chain is said to be **irreducible** if its state space is a single communicating class; in other words, if it is possible to get to any state from any state.

2: Aperiodic: A Markov chain is aperiodic if every state is aperiodic.
An irreducible Markov chain only needs one aperiodic state to imply all states are aperiodic.

BUT let's not forget ??? ***(dangling nodes)***



Background/teleportation transition matrix

$$0.15 \begin{bmatrix} 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{bmatrix} + 0.85 \begin{bmatrix} 0 & 1/3 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 1/2 \\ 1/3 & 1/3 & 0 & 0 \end{bmatrix}$$

Graph transition matrix

teleportation

dangling node mass

mass from neighbors

$$P = \alpha \times \frac{1}{|G|} + (1-\alpha)\left(\frac{m}{|G|} + P'\right) \text{ where } \alpha = 0.15$$

Distributed PageRank

- How do we distribute the PageRank calculation of a big graph across a cluster of computers using a MapReduce framework?
- The update equation can be calculated NODE BY NODE:

$$PR(x \mid W) = \alpha\left(\frac{1}{|W|}\right) + (1-\alpha)\sum_{i=1}^{n}\frac{PR(t_i)}{C(t_i)}$$

Updated Pagerank value

Divide and conquer around each node

- Atomic level of operation for PageRank is the graph node
- Where the *Key* is node;
- And the *Value* is
    i. outlink neighbors (partial graph structure) and
    ii. PageRank of that node
- Divide the node's pagerank calculation into pieces
- And synchronize around the node by gathering the component pieces and summing them up

MR for distributed PageRank

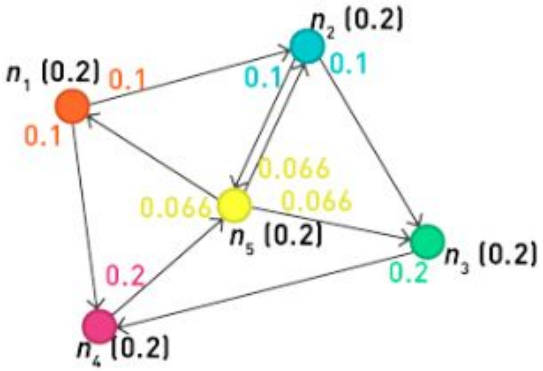
- **Initialize:** Assign an initial state vector x with the value 1/n for each node

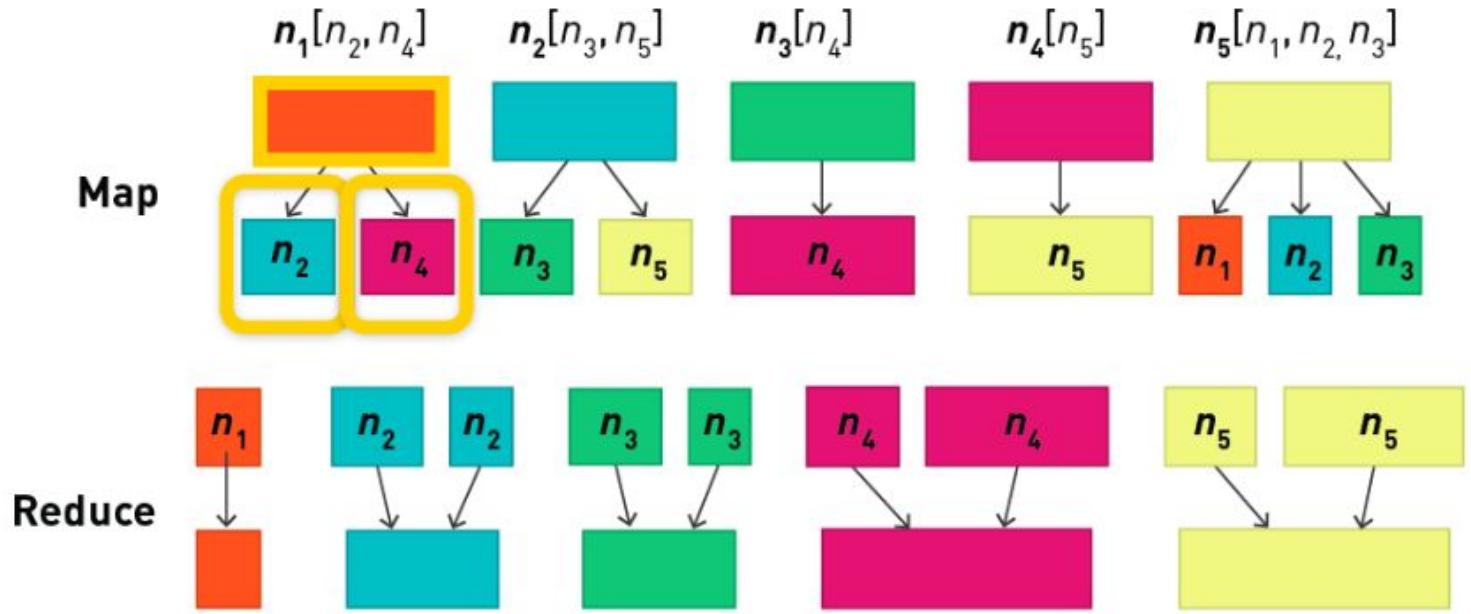
- **Mapper:** Each input node needs to distribute its PageRank probability mass to its outlink neighbors thereby generating multiple records in the output
  - Yields: Graph structure, and outbound partial $PR(x_i)/C(t_i)$ weights

- **Reducer:** Reducer will group the generated records and sum up the component contributions from each inlink neighbor
  - Aggregates $PR(x_i)/C(t_i)$ weights, with Graph structure
  - Mixes the sum with the $1/|W|$

Mapper Step for Node N_1

$n_1: pr(n_1) \rightarrow n_2, n_4$
Mapper emits the following:
$n_2: pr(n_1)/2$
$n_4: pr(n_1)/2$
$n_1 \rightarrow n_2, n_4$

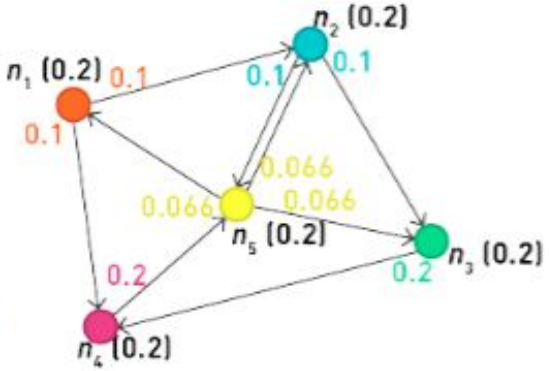


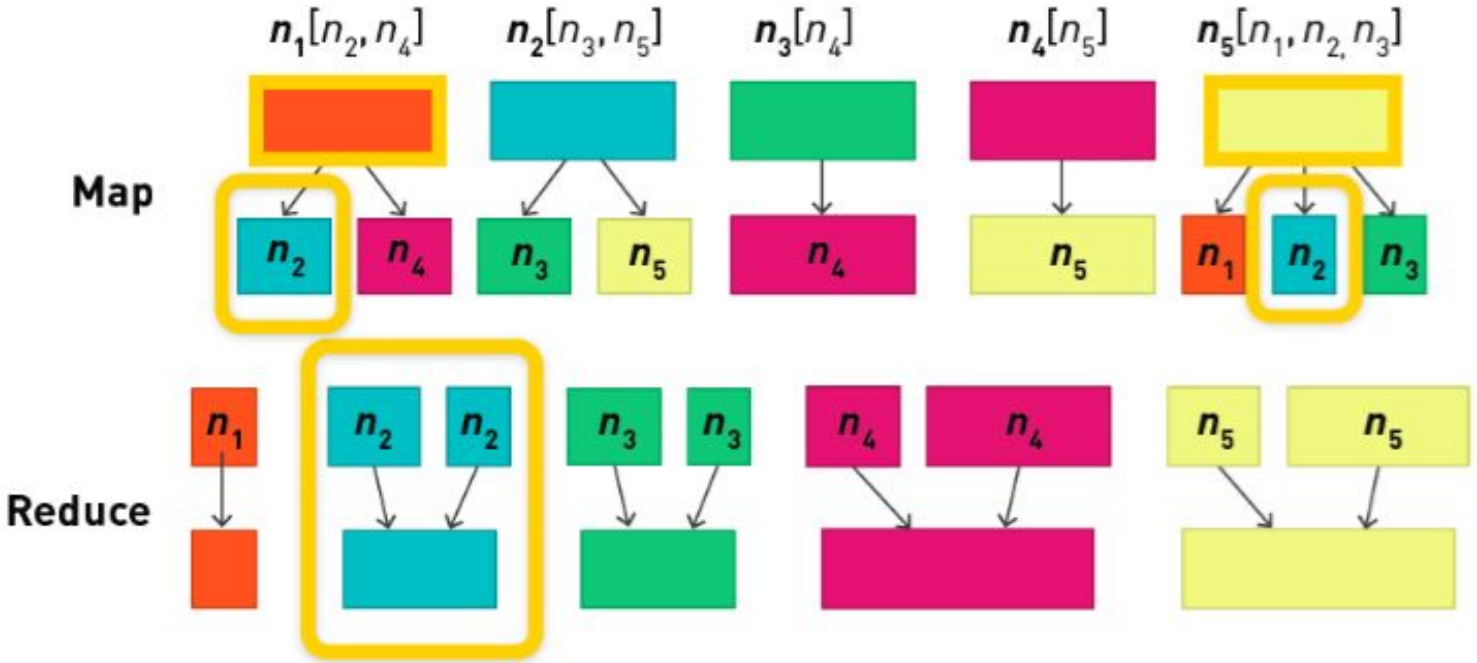


Reducer Step for Node N_1

Reducer Stream
n_1:.....
n_1:.....
$n_2: pr(n_1)/2$
$n_2: pr(n_5)/3$
$n_2 \rightarrow n_3, n_5$

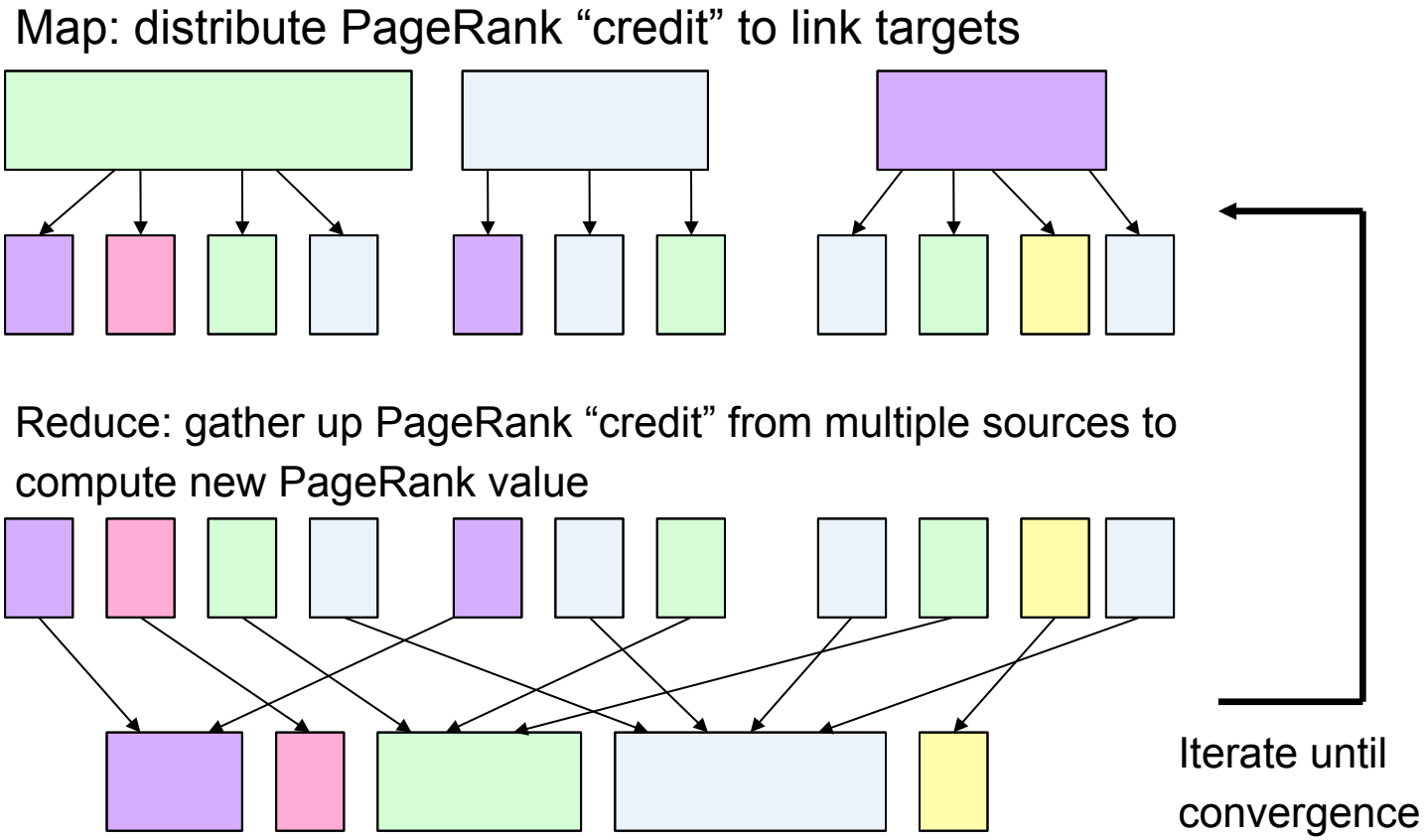
Join these records for n2
Generating a new record node for n2
$n_2: pr(n_2) \rightarrow n_2, n_5$





PageRank in MapReduce

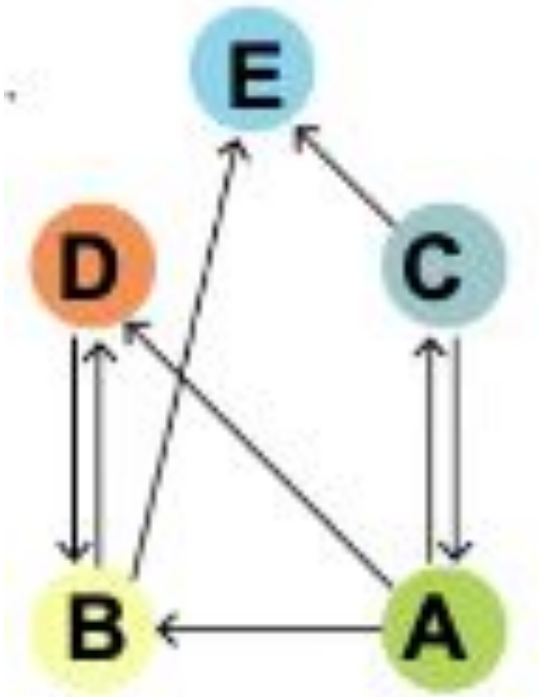
Like a Sparse Matrix by Vector multiplication



More detail: http://www.candpgeneration.com/261/PageRank-illustration.pdf

Graph: with initial pagerank value of 0.2

- Calculate the PageRank of node A after one iteration of pagerank



$$P = \alpha \times \frac{1}{|G|} + (1-\alpha)\left(\frac{m}{|G|} + P'\right) \text{ where } \alpha = 0.15$$

Full Pagerank: teleportation + dangling node treatment

Part 1

Map: distribute PageRank "credit" to link targets, and pass the graph structure

{outlinks}

A	B	C	D	A
0.2	0.2/3	0.2/3	0.2/3	-
{B,C,D}	{}	{}	{}	{B,C,D}

B	D	E	B
0.2	0.2/2	0.2/2	-
{D,E}	{}	{}	{D,E}

C	A	E	C
0.2	0.2/2	0.2/2	-
{A,E}	{}	{}	{A,E}

D	B	D
0.2	0.2/1	-
{B}	{}	{B}

E	E
0.2	-
{}	{}

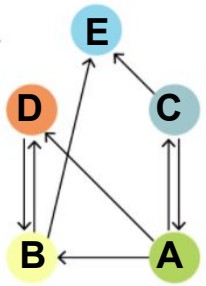
Reduce: gather up PageRank "credit" from multiple sources to compute new PageRank value

A	B	C	D	E
0.149	0.291	0.121	0.205	0.234
{B,C,D}	{D,E}	{A,E}	{B}	{}

Iterate until convergence

Part 2

Dangling Node!! Its mass has nowhere to go.



distribute uniformly: 0.2/5 where 5 is the number of nodes

Dangling Mass 0.2

0.149 = 0.15*(1/5) + (1-0.15)*(0.2/5 + 0.2/2)

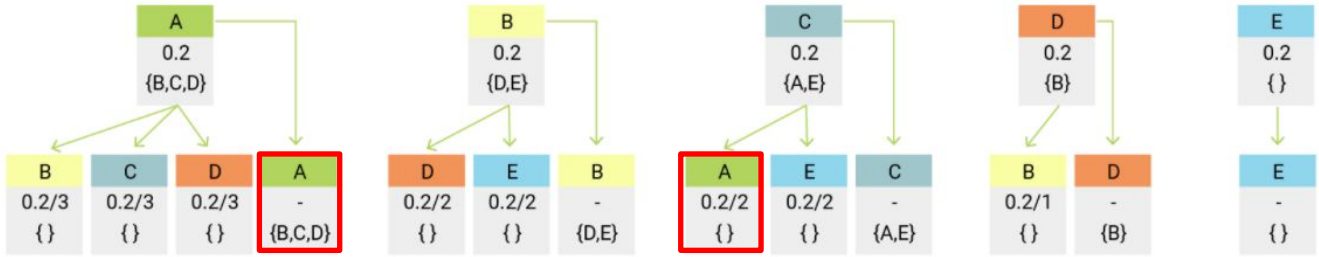
Sanity check -> 0.149 + 0.291 + 0.121 + 0.205 + 0.234 = 1.0

$$P = \alpha \times \frac{1}{|G|} + (1-\alpha)\left(\frac{m}{|G|} + P'\right) \text{ where } \alpha = 0.15$$

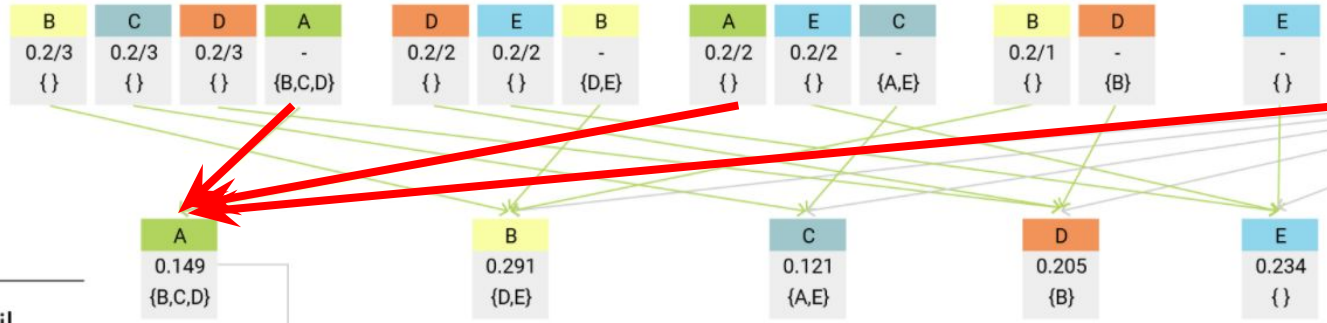
Full Pagerank: teleportation + dangling node treatment

Part 1

Map: distribute PageRank "credit" to link targets, and pass the graph structure



Reduce: gather up PageRank "credit" from multiple sources to compute new PageRank value



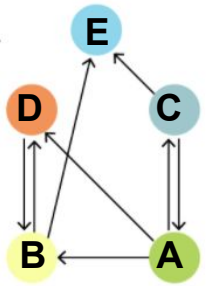
Iterate until convergence

0.149 = 0.15*(1/5) + (1-0.15)*(0.2/5 + 0.2/2)

Sanity check -> 0.149 + 0.291 + 0.121 + 0.205 + 0.234 = 1.0

Part 2

Dangling Node!! Its mass has nowhere to go.

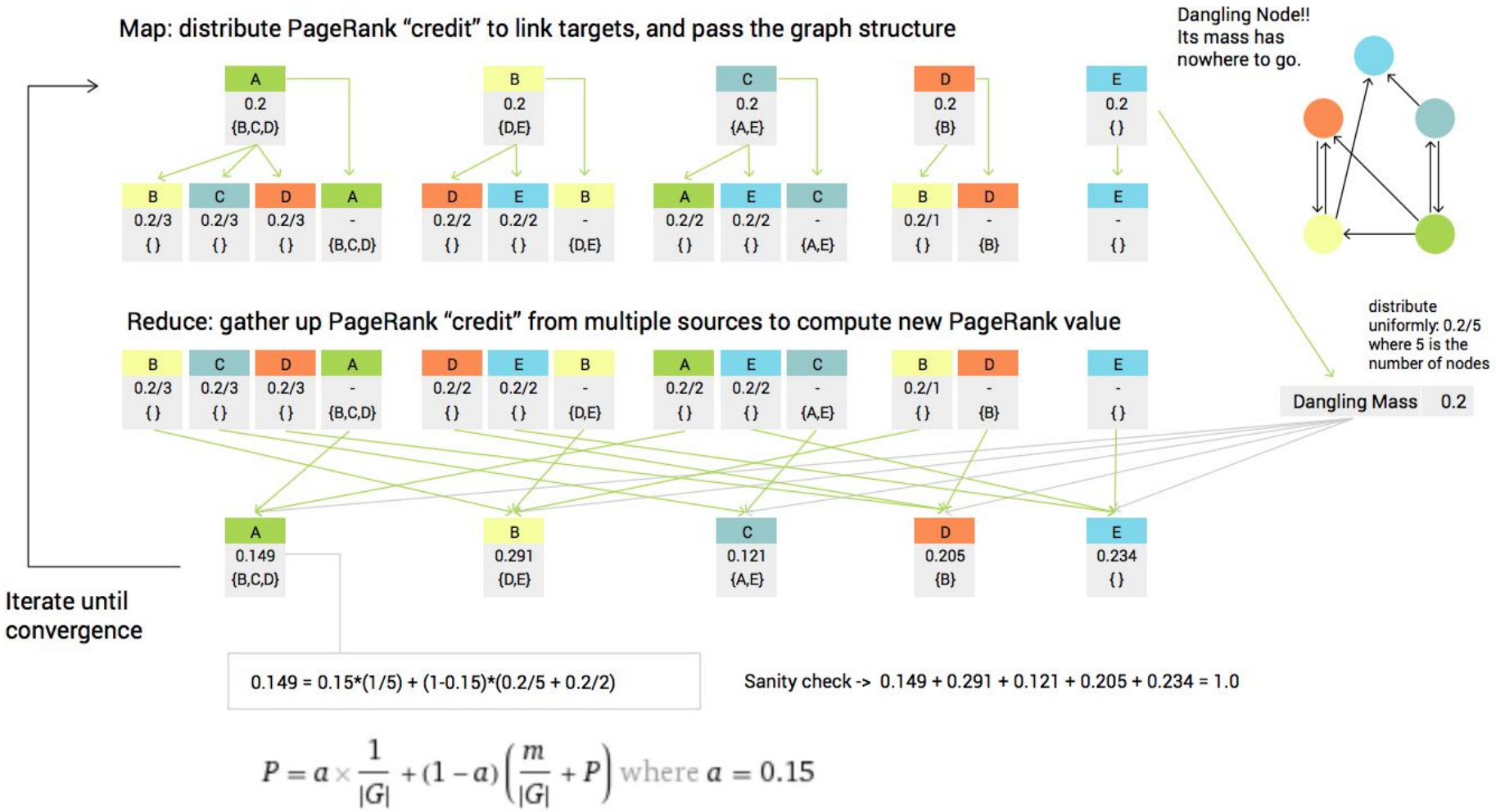


distribute uniformly: 0.2/5 where 5 is the number of nodes

Dangling Mass 0.2

$$P = \alpha \times \frac{1}{|G|} + (1-\alpha)\left(\frac{m}{|G|} + P'\right) \text{ where } \alpha = 0.15$$

PageRank in MapReduce - Illustrated



$$P = a \times \frac{1}{|G|} + (1-a)\left(\frac{m}{|G|} + P\right) \text{ where } a = 0.15$$

PageRank homegrown

```
def computeContribs(urls, rank):
    num_urls = len(urls)
    for url in urls:
        yield (url, rank / num_urls)

def parseDataGraph(line):
    fields = line.split(':')
    return(fields[0], fields[1].split(','))

links = sc.textFile("PageRank.txt").map(parseDataGraph).cache()
ranks = links.map(lambda (url, neighbors): (url, 1.0))
for iteration in xrange(10):
    contribs = links.join(ranks).flatMap(lambda (url, (urls, rank)): computeContribs(urls, rank))
    ranks = contribs.reduceByKey(lambda x,y: x + y).mapValues(lambda rank: rank * 0.85 + 0.15)
print ranks.collect()
sc.stop()
```

^^ Naive example:
https://github.com/apache/spark/blob/master/examples/src/main/python/pagerank.py

GraphX example:
https://spark.apache.org/docs/latest/graphx-programming-guide.html#pagerank
GraphX implementation:
https://github.com/apache/spark/blob/master/graphx/src/main/scala/org/apache/spark/graphx/lib/PageRank.scala

PageRank homegrown

$$PR(x^{(k+1)}|W) = P^T x^{(k)} = x^{(k)}((1-\alpha)H + \alpha T)$$

```
def computeContribs(urls, rank):
    num_urls = len(urls)
    for url in urls:
        yield (url, rank / num_urls)

def parseDataGraph(line):
    fields = line.split(':')
    return(fields[0], fields[1].split(','))

links = sc.textFile("PageRank.txt").map(parseDataGraph).cache()
ranks = links.map(lambda (url, neighbors): (url, 1.0))
for iteration in xrange(10):
    contribs = links.join(ranks).flatMap(lambda (url, (urls, rank)): computeContribs(urls, rank))
    ranks = contribs.reduceByKey(lambda x,y: x + y).mapValues(lambda rank: rank * 0.85 + 0.15)
print ranks.collect()
sc.stop()
```

join

flatMap

Example:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

$$\alpha = 0.15$$

$$P = \alpha \times \frac{1}{|G|} + (1-\alpha)($$

Links

NodeID	Neighbors (outlinks)
1	2, 3
2	3, 4
3	5
4	6
5	1, 4
6	5

Ranks

NodeID	Rank ratings
1	1.0
2	1.0
3	1.0
4	1.0
5	1.0
6	1.0

Join

Joined table

NodeID	Neighbors	Rank Ratings
1	2, 3	1.0
2	3, 4	1.0
3	5	1.0
4	6	1.0
5	1, 4	1.0
6	5	1.0

flat Map

Contribs

NeighborId	Rank Ratings
2	0.5
3	0.5
3	0.5
4	0.5
5	1.0
6	1.0
1	0.5
4	0.5
5	1.0

PageRank homegrow

p = pH + teleportation
where H is a mixture of the teleportation factor + prob
d is the mixing coefficient
1/N

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

```
def computeContribs(urls, rank):
    num_urls = len(urls)
    for url in urls:
        yield (url, rank / num_urls)

def parseDataGraph(line):
    fields = line.split(':')
    return(fields[0], fields[1].split(','))

links = sc.textFile("PageRank.txt").map(parseDataGraph).cache()
ranks = links.map(lambda (url, neighbors): (url, 1.0))
for iteration in xrange(10):
    contribs = links.join(ranks).flatMap(lambda (url, (urls, rank)): computeContribs(urls, rank))
    ranks = contribs.reduceByKey(lambda x,y: x + y).mapValues(lambda rank: rank * 0.85 + 0.15)
print ranks.collect()
sc.stop()
```

reduceByKey

mapValues

teleportation parameter
d is 0.85

Teleportation bug?

Contribs

NeighborId	Rank Ratings
2	0.5
3	0.5
3	0.5
4	0.5
5	1.0
6	1.0
1	0.5
4	0.5
5	1.0

reduce ByKey

NodeID	Rank ratings
1	0.5
2	0.5
3	1.0
4	1.0
5	2.0
6	1.0

mapValues

Updated Ranks

NodeID	Rank ratings
1	0.575
2	0.575
3	1.0
4	1.0
5	1.85
6	1.0

PageRank homegrow

p = pH + teleportation
where H is a mixture of the teleportation factor + prob
d is the mixing coefficient
1/N

$$p' = \alpha\left(\frac{1}{|G|}\right) + (1-\alpha)\left(\frac{m}{|G|} + p\right)$$

```
def computeContribs(urls, rank):
    num_urls = len(urls)
    for url in urls:
        yield (url, rank / num_urls)

def parseDataGraph(line):
    fields = line.split(':')
    return(fields[0], fields[1].split(','))

links = sc.textFile("PageRank.txt").map(parseDataGraph).cache()
ranks = links.map(lambda (url, neighbors): (url, 1.0))
for iteration in xrange(10):
    contribs = links.join(ranks).flatMap(lambda (url, (urls, rank)): computeContribs(urls, rank))
    ranks = contribs.reduceByKey(lambda x,y: x + y).mapValues(lambda rank: rank * 0.85 + 0.15)
print ranks.collect()
sc.stop()
```

reduceByKey

mapValues

teleportation parameter
d is 0.85

Teleportation bug?

Challenge: modify code to handle dangling node mass, m

$$p' = \alpha\left(\frac{1}{|G|}\right) + (1-\alpha)\left(\frac{m}{|G|} + p\right)$$

Contribs

NeighborId	Rank Ratings
2	0.5
3	0.5
3	0.5
4	0.5
5	1.0
6	1.0
1	0.5
4	0.5
5	1.0

re
ByKey

4	1.0
5	2.0
6	1.0

s

Updated Ranks

odeID	Rank ratings
1	0.575
2	0.575
3	1.0
4	1.0
5	1.85
6	1.0

Accumulator with Example

- Accumulators are write-only and initialize once variables where only tasks that are running on workers are allowed to update and updates from the workers get propagated automatically to the driver program.
- But, only the driver program is allowed to access the Accumulator variable using the value property.

```
accum=sc.accumulator(0)
rdd=spark.sparkContext.parallelize([1,2,3,4,5])
rdd.foreach(lambda x:accum.add(x))
print(accum.value) #Accessed by driver
```

Here, we have created an accumulator variable accum using spark.sparkContext.accumulator(0) with initial value 0. Later, we are iterating each element in an rdd using foreach() action and adding each element of rdd to accum variable. Finally, we are getting accumulator value using accum.value property.

Note that, In this example, `rdd.foreach()` is executed on workers and `accum.value` is called from PySpark driver program.

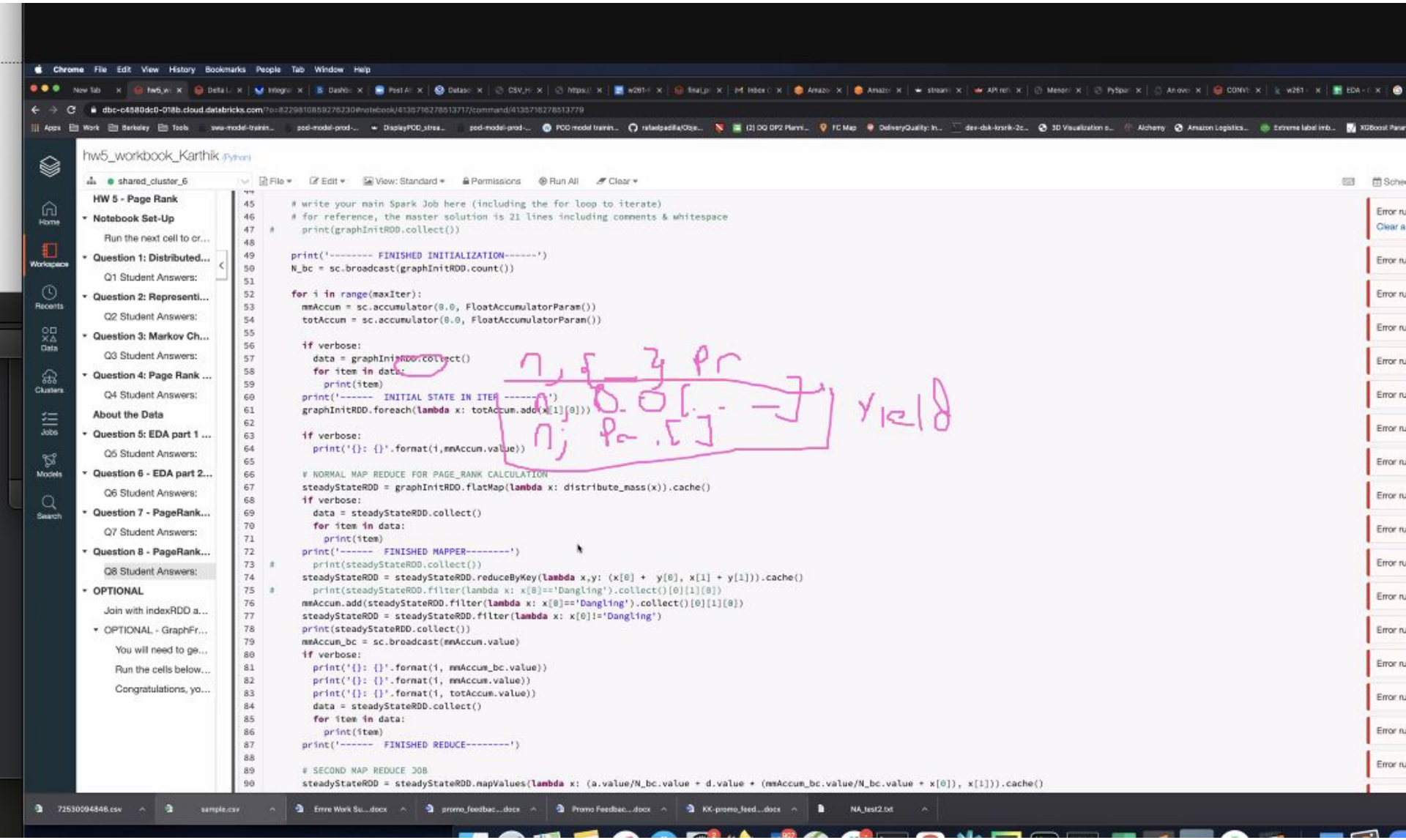
Let's see another example of an accumulator, this time will do with a function.

An alternative solution (join)

hw5_workbook_Karthik (Python)

shared_cluster_6 | File | Edit | View: Standard | Permissions | Run All | Clear


```
40          for item in adj_list:
41            yield(item[0], (page_rank*item[1]/total_weight,[]))
42        else:
43          yield('Dangling', (page_rank, []))
44
45      # write your main Spark Job here (including the for loop to iterate)
46      # for reference, the master solution is 21 lines including comments & whitespace
47  #     print(graphInitRDD.collect())
48
49      print('-------- FINISHED INITIALIZATION------')
50      N_bc = sc.broadcast(graphInitRDD.count())
51
52      for i in range(maxIter):
53        mmAccum = sc.accumulator(0.0, FloatAccumulatorParam())
54        totAccum = sc.accumulator(0.0, FloatAccumulatorParam())
55
56        if verbose:
57          data = graphInitRDD.collect()
58          for item in data:
59            print(item)
60        print('------  INITIAL STATE IN ITER --------')
61        graphInitRDD.foreach(lambda x: totAccum.add(x[1][0]))
62
63        if verbose:
64          print('{}: {}'.format(i,mmAccum.value))
65
66        # NORMAL MAP REDUCE FOR PAGE_RANK CALCULATION
67        steadyStateRDD = graphInitRDD.flatMap(lambda x: distribute_mass(x)).cache()
68        if verbose:
69          data = steadyStateRDD.collect()
70          for item in data:
71            print(item)
72        print('------  FINISHED MAPPER--------')
73  #       print(steadyStateRDD.collect())
74        steadyStateRDD = steadyStateRDD.reduceByKey(lambda x,y: (x[0] +  y[0], x[1] + y[1])).cache()
75  #       print(steadyStateRDD.filter(lambda x: x[0]=='Dangling').collect()[0][1][0])
76        mmAccum.add(steadyStateRDD.filter(lambda x: x[0]=='Dangling').collect()[0][1][0])
77        steadyStateRDD = steadyStateRDD.filter(lambda x: x[0]!='Dangling')
78        print(steadyStateRDD.collect())
79        mmAccum_bc = sc.broadcast(mmAccum.value)
80        if verbose:
81          print('{}: {}'.format(i, mmAccum_bc.value))
82          print('{}: {}'.format(i, mmAccum.value))
83          print('{}: {}'.format(i, totAccum.value))
84          data = steadyStateRDD.collect()
85          for item in data:
86            print(item)
```

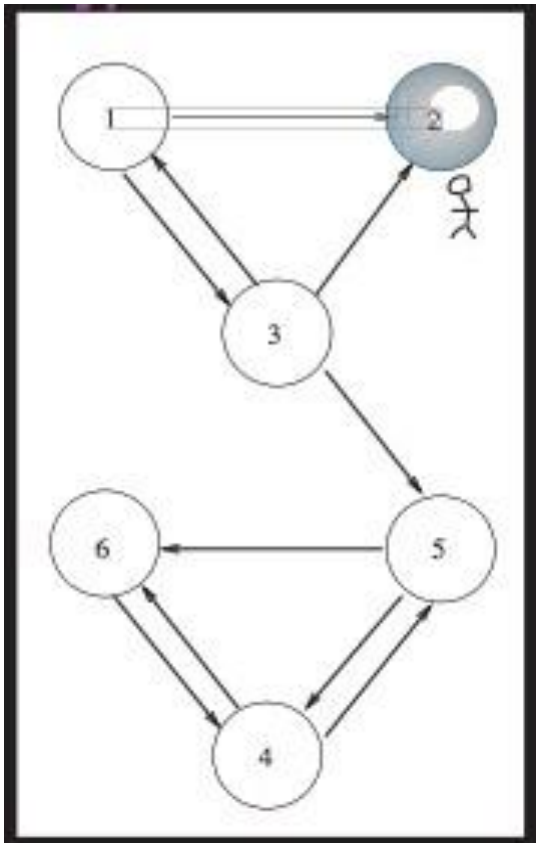

hw5_workbook_Karthik (Python)
shared_cluster_6
File
Edit
View: Standard
Permissions
Run All
Clear
HW 5 - Page Rank
Notebook Set-Up
Run the next cell to cr...
Question 1: Distributed...
Q1 Student Answers:
Question 2: Representi...
Q2 Student Answers:
Question 3: Markov Ch...
Q3 Student Answers:
Question 4: Page Rank ...
Q4 Student Answers:
About the Data
Question 5: EDA part 1 ...
Q5 Student Answers:
Question 6 - EDA part 2...
Q6 Student Answers:
Question 7 - PageRank...
Q7 Student Answers:
Question 8 - PageRank...
Q8 Student Answers:
OPTIONAL
Join with indexRDD a...
OPTIONAL - GraphFr...
You will need to ge...
Run the cells below...
Congratulations, yo...
# write your main Spark Job here (including the for loop to iterate)
# for reference, the master solution is 21 lines including comments & whitespace
# print(graphInitRDD.collect())
print('-------- FINISHED INITIALIZATION------')
N_bc = sc.broadcast(graphInitRDD.count())
for i in range(maxIter):
mmAccum = sc.accumulator(0.0, FloatAccumulatorParam())
totAccum = sc.accumulator(0.0, FloatAccumulatorParam())
if verbose:
data = graphInitRDD.collect()
for item in data:
print(item)
print('------ INITIAL STATE IN ITER -------')
graphInitRDD.foreach(lambda x: totAccum.add(x[1][0]))
if verbose:
print('{}: {}'.format(i,mmAccum.value))
# NORMAL MAP REDUCE FOR PAGE_RANK CALCULATION
steadyStateRDD = graphInitRDD.flatMap(lambda x: distribute_mass(x)).cache()
if verbose:
data = steadyStateRDD.collect()
for item in data:
print(item)
print('------ FINISHED MAPPER--------')
# print(steadyStateRDD.collect())
steadyStateRDD = steadyStateRDD.reduceByKey(lambda x,y: (x[0] + y[0], x[1] + y[1])).cache()
# print(steadyStateRDD.filter(lambda x: x[0]=='Dangling').collect()[0][1][0])
mmAccum.add(steadyStateRDD.filter(lambda x: x[0]=='Dangling').collect()[0][1][0])
steadyStateRDD = steadyStateRDD.filter(lambda x: x[0]!='Dangling')
print(steadyStateRDD.collect())
mmAccum_bc = sc.broadcast(mmAccum.value)
if verbose:
print('{}: {}'.format(i, mmAccum_bc.value))
print('{}: {}'.format(i, mmAccum.value))
print('{}: {}'.format(i, totAccum.value))
data = steadyStateRDD.collect()
for item in data:
print(item)
print('------ FINISHED REDUCE--------')
# SECOND MAP REDUCE JOB
steadyStateRDD = steadyStateRDD.mapValues(lambda x: (a.value/N_bc.value + d.value + (mmAccum_bc.value/N_bc.value + x[0]), x[1])).cache()
yield

Live Session Outline

- **Housekeeping**
  - Where we are in the course
  - Last Week Review
- **Page Rank**
  - Adjacency Matrices and Markov Processes
  - Calculating Steady State and The Power Method
  - Distributed PageRank
  - Dangling Nodes
  - Topic Specific PageRank
- **Bonus [Optional] Topics**
  - Contextual advertising [Optional]
  - Text as graph: TextRank [Optional]
    - Keyword extraction (from text/target pages)
    - Text Summarization

Dangling nodes (deadend nodes)

- **Node 2 is an example of a dangling node (it has no outlinks)**



Question:
what other problem can you see in this graph?

More complete PageRank in MR

- **Random jump factor (teleportation)**
- **Dangling nodes:**
  - Dangling nodes are nodes in the graph that have no outgoing edges (dead end nodes)i.e., their adjacency lists are empty.
  - In the hyperlink graph of the web, these might correspond to pages in a crawl that have not been downloaded yet.
  - If we simply run the algorithm presented above on graphs with dangling nodes, the total PageRank mass will not be conserved, since no key-value pairs will be emitted when a dangling node is encountered in the mappers
  - (could end up with lot of missing mass from the initialization phase)

Get the mapper to track the PR Mass of dangling nodes in Hadoop Counter (Spark accumulator) and redistribute to all nodes

- **The proper treatment of PageRank mass “lost" at the dangling nodes is to redistribute it across all nodes in the graph evenly**
- Instrument MR with counters
  - One simple approach is by instrumenting the algorithm presented above with counters: whenever the mapper processes a node with an empty adjacency list, it keeps track of the node's PageRank value in the counter.
  - At the end of the iteration, we can access the counter to find out how much PageRank mass was lost at the dangling nodes.
  - NOTE In Hadoop, counters are 8-byte integers: a simple workaround is to multiply PageRank values by a large constant, and then cast as an integer.

How to redistribute the dangling PR Mass?

- Either way, we arrive at the amount of PageRank mass lost at the dangling nodes. This then must be redistributed evenly across all nodes.
- This redistribution process can be accomplished by mapping over all nodes again.
- At the same time, we can take into account the random jump factor. For each node, its current PageRank value p is updated to the final PageRank value p0 according to the following formula:

$$p' = \alpha\left(\frac{1}{|G|}\right) + (1-\alpha)\left(\frac{m}{|G|} + p\right)$$

  - where m is the missing PageRank mass, and |G| is the number of nodes in the entire graph.
- Need a second MR job with a Mapper to do this

Each PageRank iteration requires 2 MR jobs

- **Distribute PageRank mass along graph edges,**
- **And the second to take care of dangling nodes and the random jump factor.**
- **NOTE:**
  - At end of each iteration, we end up with exactly the same data structure as the beginning, which is a requirement for the iterative algorithm to work.
  - Also, the PageRank values of all nodes sum up to one, which ensures a valid probability distribution.

How many Iterations for MR PageRank?

- Typically, PageRank is iterated until convergence, i.e., when the PageRank values of nodes no longer change (within some tolerance, to take into account, for example, floating point precision errors).

- Stopping Criteria

Alternative stopping Criteria?

- **PageRank values of nodes**
  - Typically, PageRank is iterated until convergence, i.e., when the PageRank values of nodes no longer change (within some tolerance, to take into account, for example, floating point precision errors).
- **Fixed Number of iterations**
- **Ranks of PageRank values no longer change**
  - Alternative stopping criteria include running a fixed number of iterations (useful if one wishes to bound algorithm running time) or stopping when the ranks of PageRank values no longer change.
  - The latter is useful for some applications that only care about comparing the PageRank of two arbitrary pages and do not need the actual PageRank values.
- **Rank stability is obtained faster than the actual convergence of values.**

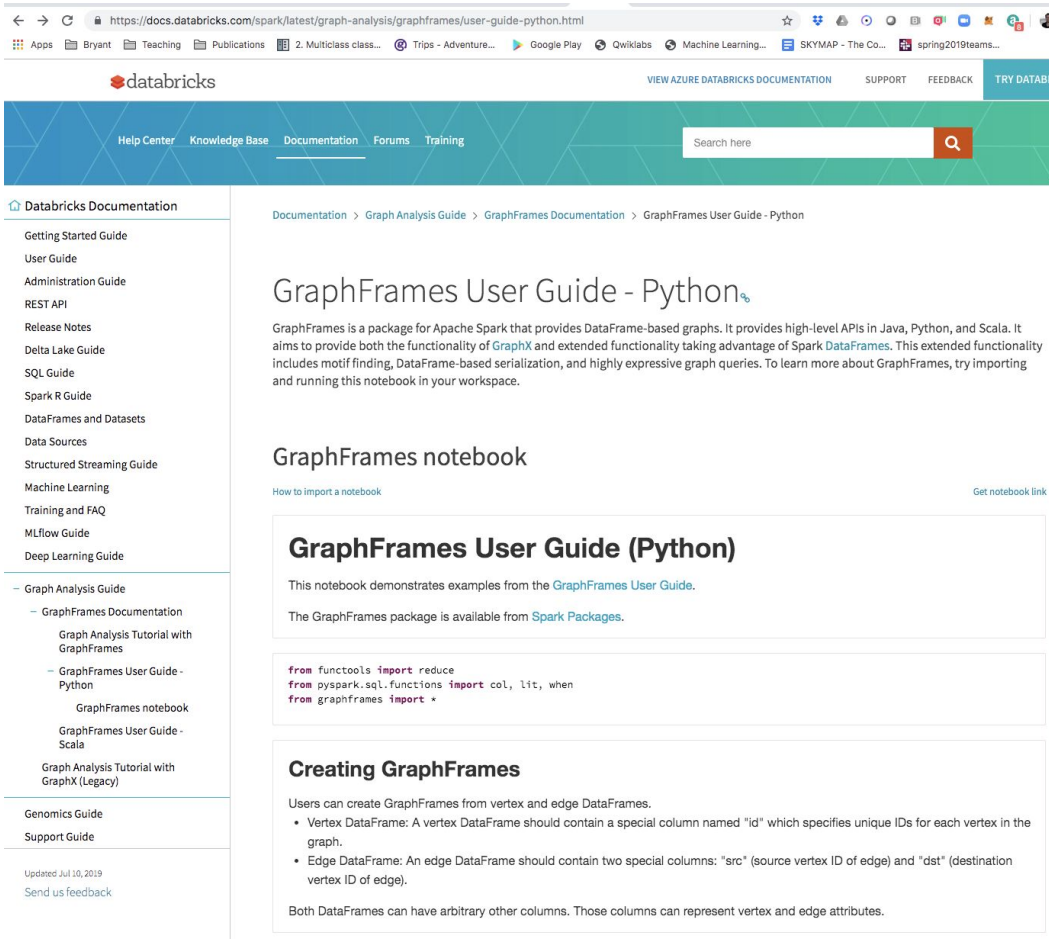
Spark GraphFrames library

- GraphFrames is a package for Apache Spark that provides DataFrame-based graphs. It provides high-level APIs in Java, Python, and Scala. It aims to provide both the functionality of GraphX and extended functionality taking advantage of Spark DataFrames. This extended functionality includes motif finding, DataFrame-based serialization, and highly expressive graph queries.
- The GraphFrames package is available from Spark Packages.
- This article demonstrates examples from the GraphFrames User Guide.
- **NOTE: Benchmark your code using the Spark GraphFrames library**

Benchmark your code using the Spark GraphFrames library

Spark GraphFrames Outline

- How create a graph in Graphframes
- Graph-based EDA
- Standard graph algorithms



https://docs.databricks.com/spark/latest/graph-analysis/graphframes/user-guide-python.html

https://docs.databricks.com/spark/latest/graph-analysis/graphframes/user-guide-python.html

Create the vertices first:

```
vertices = sqlContext.createDataFrame([
  ("a", "Alice", 34),
  ("b", "Bob", 36),
  ("c", "Charlie", 30),
  ("d", "David", 29),
  ("e", "Esther", 32),
  ("f", "Fanny", 36),
  ("g", "Gabby", 60)], ["id", "name", "age"])
```

And then some edges:

```
edges = sqlContext.createDataFrame([
  ("a", "b", "friend"),
  ("b", "c", "follow"),
  ("c", "b", "follow"),
  ("f", "c", "follow"),
  ("e", "f", "follow"),
  ("e", "d", "friend"),
  ("d", "a", "friend"),
  ("a", "e", "friend")
], ["src", "dst", "relationship"])
```

Let's create a graph from these vertices and these edges:

```
g = GraphFrame(vertices, edges)
print(g)
```

```
GraphFrame(v:[id: string, name: string ... 1 more field], e:[src: string, dst: string ... 1 more field])
```

```
# This example graph also comes with the GraphFrames package.
from graphframes.examples import Graphs
same_g = Graphs(sqlContext).friends()
print(same_g)
```

```
GraphFrame(v:[id: string, name: string ... 1 more field], e:[src: string, dst: string ... 1 more field])
```

Basic graph and DataFrame queries

GraphFrames provide several simple graph queries, such as node degree.

Also, since GraphFrames represent graphs as pairs of vertex and edge DataFrames, it is easy to make powerful queries directly on the vertex and edge DataFrames. Those DataFrames are made available as vertices and edges fields in the GraphFrame.

```
display(g.vertices)
```

id	name	age
a	Alice	34
b	Bob	36
c	Charlie	30
d	David	29
e	Esther	32
f	Fanny	36
g	Gabby	60

```
display(g.edges)
```

src	dst	relationship
a	b	friend
b	c	follow
c	b	follow

The incoming degree of the vertices:

```
display(g.inDegrees)
```

id	inDegree
f	1
e	1
d	1
c	2
b	2
a	1

The outgoing degree of the vertices:

```
display(g.outDegrees)
```

id	outDegree
f	1
e	2

You can run queries directly on the vertices DataFrame. For example, we can find the age of the youngest person in the graph:

```
youngest = g.vertices.groupBy().min("age")
display(youngest)
```

min(age)
29

Likewise, you can run queries on the edges DataFrame. For example, let's count the number of 'follow' relationships in the graph:

```
numFollows = g.edges.filter("relationship = 'follow'").count()
print("The number of follow edges is", numFollows)
```

```
The number of follow edges is 4
```

Motif finding

Using motifs you can build more complex relationships involving edges and vertices. The following cell finds the pairs of vertices with edges in both directions between them. The result is a DataFrame, in which the column names are given by the motif keys.

Check out the GraphFrame User Guide for more details on the API.

```
# Search for pairs of vertices with edges in both directions between them.
motifs = g.find("(a)-[e]->(b); (b)-[e2]->(a)")
display(motifs)
```

a	e	b	e2
{"id":"c","name":"Charlie","age":"30"}	{"src":"c","dst":"b","relationship":"follow"}	{"id":"b","name":"Bob","age":"36"}	{"src":"b"
{"id":"b","name":"Bob","age":"36"}	{"src":"b","dst":"c","relationship":"follow"}	{"id":"c","name":"Charlie","age":"30"}	{"src":"c"

Since the result is a DataFrame, more complex queries can be built on top of the motif. Let us find all the reciprocal relationships in which one person is older than 30:

```
filtered = motifs.filter("b.age > 30 or a.age > 30")
display(filtered)
```

a	e	b	e2
{"id":"c","name":"Charlie","age":"30"}	{"src":"c","dst":"b","relationship":"follow"}	{"id":"b","name":"Bob","age":"36"}	{"src":"b"
{"id":"b","name":"Bob","age":"36"}	{"src":"b","dst":"c","relationship":"follow"}	{"id":"c","name":"Charlie","age":"30"}	{"src":"c"

Standard graph algorithms

GraphFrames comes with a number of standard graph algorithms built in:

- Breadth-first search (BFS)
- Connected components
- Strongly connected components
- Label Propagation Algorithm (LPA)
- PageRank (regular and personalized)
- Shortest paths
- Triangle count

Breadth-first search (BFS)

Search from "Esther" for users of age < 32.

```
paths = g.bfs("name = 'Esther'", "age < 32")
display(paths)
```

from	e0	to
{"id":"e","name":"Esther","age":"32"}	{"src":"e","dst":"d","relationship":"friend"}	{"id":"d","name":"David","age":"29"}

PageRank

Identify important vertices in a graph based on connections.

```
results = g.pageRank(resetProbability=0.15, tol=0.01)
display(results.vertices)
```

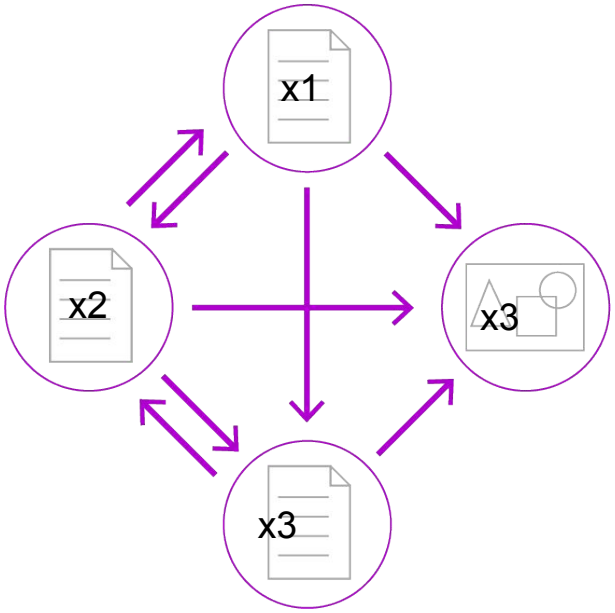
id	name	age	pagerank
g	Gabby	60	0.1799821386239711
b	Bob	36	2.655507832863289
e	Esther	32	0.37085233187676075
a	Alice	34	0.44910633706538744
f	Fanny	36	0.3283606792049851
d	David	29	0.3283606792049851
c	Charlie	30	2.6878300011606218

```
display(results.edges)
```

src	dst	relationship	weight
a	b	friend	0.5
b	c	follow	1
e	f	follow	0.5
e	d	friend	0.5
c	b	follow	1
a	e	friend	0.5
f	c	follow	1

Live Session Outline

- **Page Rank**
  - Adjacency Matrices and Markov Processes
  - Calculating Steady State and The Power Method
  - Distributed PageRank
  - Dangling Nodes
  - Topic Specific PageRank
- **Bonus [Optional] Topics**
  - Contextual advertising [Optional]
  - Text as graph: TextRank [Optional]
    - Keyword extraction (from text/target pages)
    - Text Summarization



Teleportation adjustments for PageRank

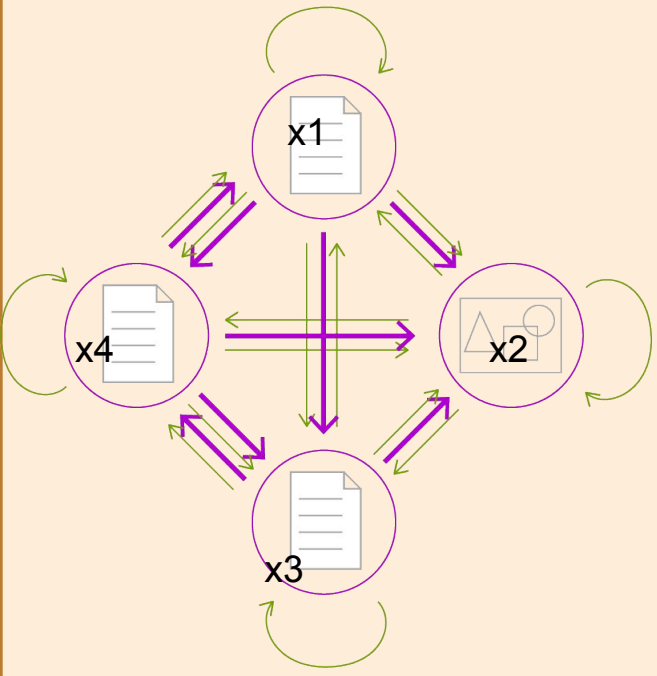
Random Jump factor adds a small amount of probability to each node to teleport to any other node.

Both issues of periodicity and irreducibility are solved at once.

1: Irreducibility/Reachable A Markov chain is said to be **irreducible** if its state space is a single communicating class; in other words, if it is possible to get to any state from any state.

2: Aperiodic: A Markov chain is aperiodic if every state is aperiodic.
An irreducible Markov chain only needs one aperiodic state to imply all states are aperiodic.

BUT let's not forget ??? ***(dangling nodes)***



Background/teleportation adjacency matrix

$$0.15 \begin{bmatrix} 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{bmatrix} + 0.85 \begin{bmatrix} 0 & 1/3 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 1/2 \\ 1/3 & 1/3 & 0 & 0 \end{bmatrix}$$

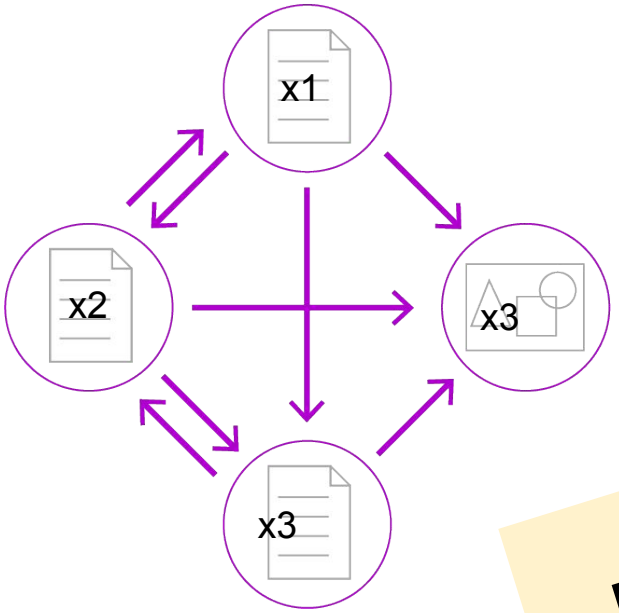
Graph adjacency matrix

teleportation

dangling node mass

mass from neighbors

$$P = \alpha \times \frac{1}{|G|} + (1-\alpha)\left(\frac{m}{|G|} + P'\right) \text{ where } \alpha = 0.15$$



Teleportation adjustments for PageRank

Random Jump factor adds a small amount of probability to each node to teleport to any other node.

Both issues of periodicity and irreducibility are solved at once.

1: Irreducibility/Reachabl

a single communic

irreducible if its state space is

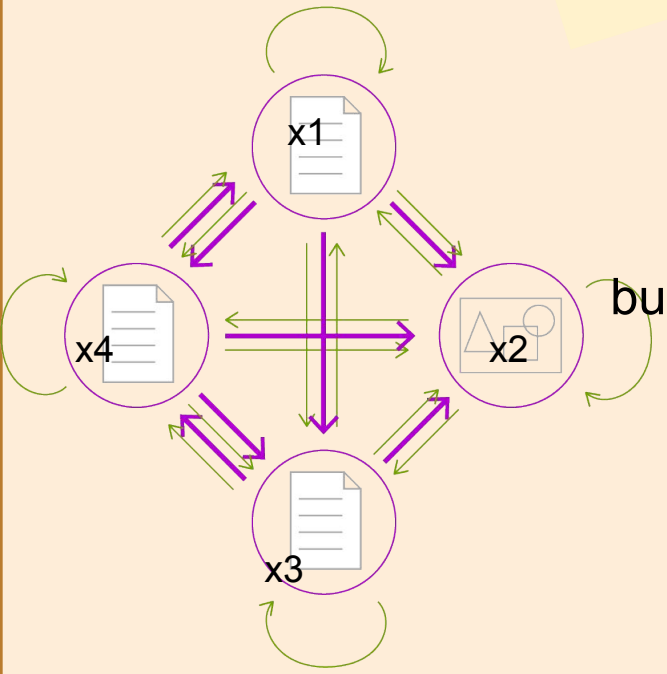
ible to get to any state from any state.

Sports-based pagerank
Bias teleportation to pages in the sports category

y state is aperiodic.

one aperiodic state to imply all states are

(dangling nodes)



Background/teleportation adjacency matrix

sports
health
sports
business

$$\begin{bmatrix} 0.4 & 1/4 & 1/4 & 1/4 \\ 0.1 & & & 1/4 \\ 0.4 & & & \\ 0.1 & 1/4 & 1/4 & 1/4 \end{bmatrix}$$

bias these teleportations in the same way

$+ \ 0.85$

Graph adjacency matrix

$$\begin{bmatrix} 0 & 1/3 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 1/2 \\ 1/3 & 1/3 & 0 & 0 \end{bmatrix}$$

biased teleportation

dangling node mass

mass from neighbors

$$P = \alpha \times \frac{1}{|G|} + (1-\alpha)\left(\frac{m}{|G|} + P'\right) \text{ where } \alpha = 0.15$$


dmoz
In partnership with
Aol.
Follow @dmoz
about dmoz | dmoz blog | suggest URL | help | link | editor login
Search
advanced
Category
Sub-Categories
Arts
Movies, Television, Music...
Business
Jobs, Real Estate, Investing...
Computers
Internet, Software, Hardware...
Games
Video Games, RPGs, Gambling...
Health
Fitness, Medicine, Alternative...
Home
Family, Consumers, Cooking...
Kids and Teens
Arts, School Time, Teen Life...
News
Media, Newspapers, Weather...
Recreation
Travel, Food, Outdoors, Humor...
Reference
Maps, Education, Libraries...
Regional
US, Canada, UK, Europe...
Science
Biology, Psychology, Physics...
Shopping
Clothing, Food, Gifts...
Society
People, Religion, Issues...
Sports
Baseball, Soccer, Basketball...
World
Català, Česky, Dansk, Deutsch, Español, Esperanto, Français, Galego, Hrvatski, Italiano, Lietuvių, Magyar, Nederlands, Norsk, Polski, Português, Română, Slovensky, Suomi, Svenska, Türkçe, Български, Ελληνικά, Русский, Українська, العربية, עברית, ไทย, 日本語, 简体中文, 繁體中文, ...
Become an Editor
Help build the largest human-edited directory of the web
Copyright © 1998-2016 AOL Inc.

Topic Specific Pagerank:

- **Implementation**
  - **offline**:Compute pagerank distributions wrt to *individual* categories
  - Query independent model as before
  - Modify teleportation matrix
    - assign teleportation probability mass to webpages in class of interest only; and zero to all non-class pages
  - Each page has multiple pagerank scores – one for each ODP category (DMOZ), with teleportation only to that category

- **MapReduce (one set for 20+1 overall PR)**
  - **Page calculation**
  - **Dangling nodes plus teleportation**

- **End of lecture**

- **Bonus [Optional] Topics**
  - Contextual advertising [Optional]
  - Text as graph: TextRank [Optional]
    - Keyword extraction (from text/target pages)
    - Text Summarization
  - Personalized PageRank (Markov models have also been used to analyze web navigation behavior of users. A user's web link transition on a particular website can be modeled using first- or second-order Markov models and can be used to make predictions regarding future navigation and to personalize the web page for an individual user.)