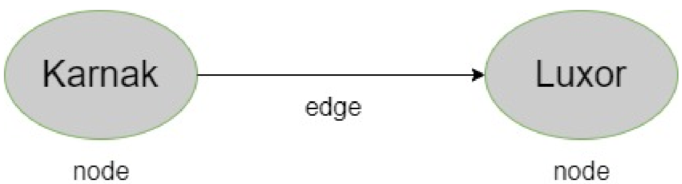
Comp 363 - Design and Analysis of Computer Algorithms

Spring Semester 2020 - Week 13 - Part 1

Dr Nick Hayward

graphs - intro recap

- graph data structure in computer science
- a way to model a given set of connections
- commonly use a *graph* to model patterns and connections for a given problem
- e.g. connections may infer relationships within data
- graph includes *nodes* and *edges*
  - *help us define such connections*
- e.g. we have two nodes with a single edge

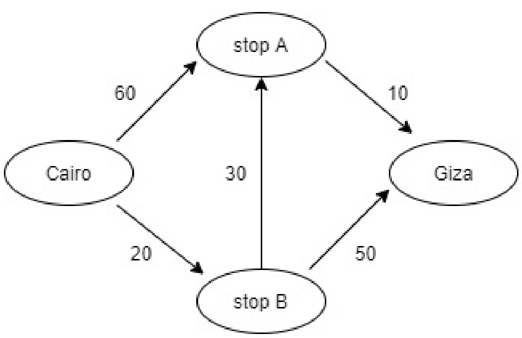


Graph Nodes and Edge

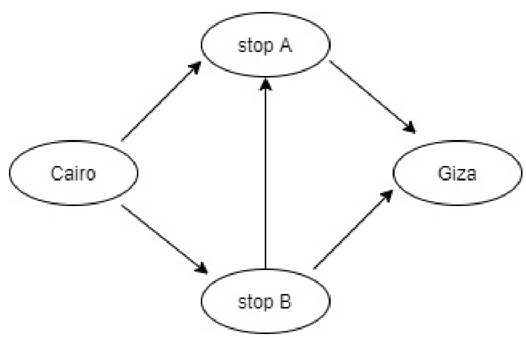
- each node may be connected to many other nodes in the graph
  - *commonly referenced as neighbour nodes*

graphs - Dijkstra's algorithm - consideration of terminology - part 1

- key concept for working with Dijkstra's algorithm is the association of values,
  - *e.g. numbers for each edge in the graph*
- valyes are the weights assigned to the edge in the graph
- when we assign weights to an edge
  - *creating a weighted graph*
- if we do not assign weights to edges
  - *defining an unweighted graph*
- e.g. weighted and unweighted graphs



Weighted



Unweighted

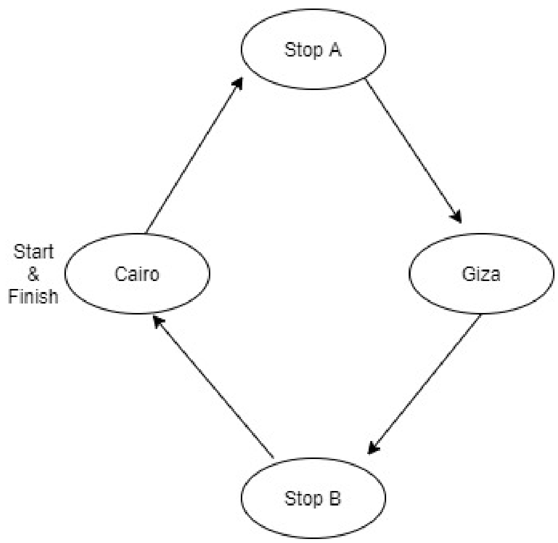
Graphs - Weighted and Unweighted

graphs - Dijkstra’s algorithm - consideration of terminology - part 2

- choice of weighted versus unweighted
  - *also affecta our choice of algorithm*
  - *and the context of its usage*
- e.g. if we want to calculate the shortest path in an *unweighted* graph
  - *we may use the breadth-first search algorithm*
- to perform a similar calculation for a *weighted* graph
  - *we may use Dijkstra’s algorithm…*

graphs - Dijkstra's algorithm - graph cycles - part 1

- may also encounter a graph with *cycles*
- i.e. we can cycle from `Stop A` back around to `Stop A`
- e.g.



Graph - Cycle

- we may start and finish at the same node in the graph

Algorithms and Data Structures

graphs - Dijkstra's algorithm - graph cycles - part 2

- if we consider a graph with a cycle segment
  - *need to calculate shortest path between two defined nodes*
- for most calculations commonly choose a path that avoids the cycle
- cycle will usually add greater weight to the calculation
- if we then follow cycle more than once
  - *simply adding extra weight to calculation for each completed cycle*
- if we consider an *undirected graph*
  - *now working with a cycle*
- connected nodes in an undirected graph point to each other
  - *effectively a cycle*
- each edge will add another cycle to an undirected graph
- *Dijkstra's* algorithm only works with *directed acyclic graphs* (DAGs)

Algorithms and Data Structures

graphs - Dijkstra's algorithm - shortest path - part 1

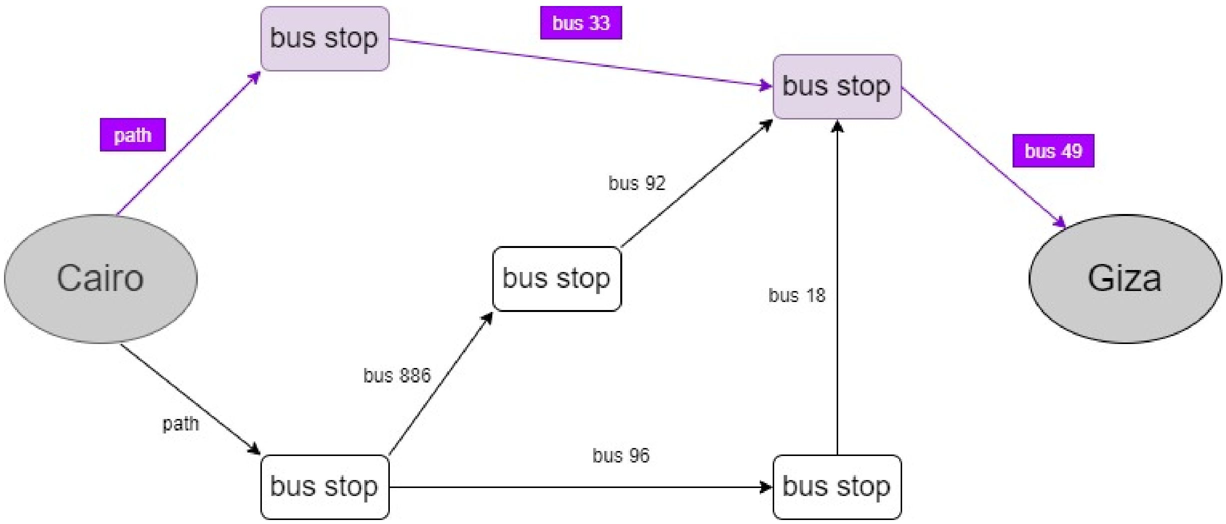
- common requirement for working with graphs
  - *a consideration of weighted and unweighted edges*
- with weighted graphs
  - *interested in options for assigning more or less weight*
  - *i.e. to edges within the graph*
- Dijkstra's algorithm helps us work with queries for paths in our graphs
- e.g. we might need to answer the question

> *which path is the shortest to a node?*

- e.g. which is the shortest path to node A?
  - *this path may not be fastest*
  - *but it will be the shortest in the graph*
- shortest because it will include least number of edges between nodes

graphs - Dijkstra's algorithm - shortest path - part 2

- e.g. we might consider the following graph for the shortest route

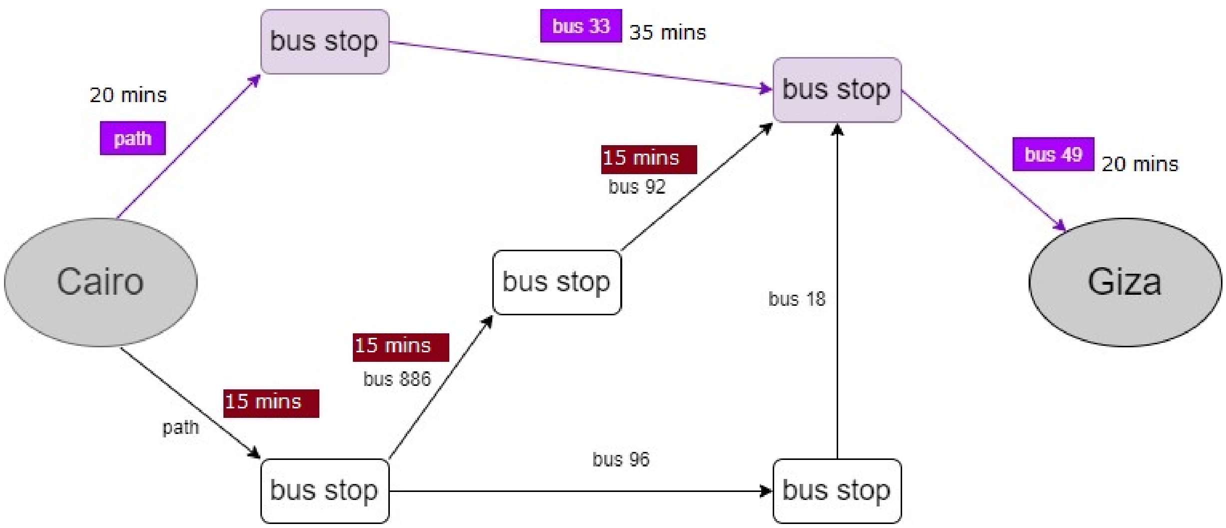


Graph Routes - shortest

- clearly see shortest path with least number of segments, three
- may use *breadth-first* search to find path with fewest segments

graphs - Dijkstra's algorithm - shortest path - part 3

- if we then added travel times to edges for competing routes
  - *i.e. costs for each edge*
- we may find a quicker path for traveling

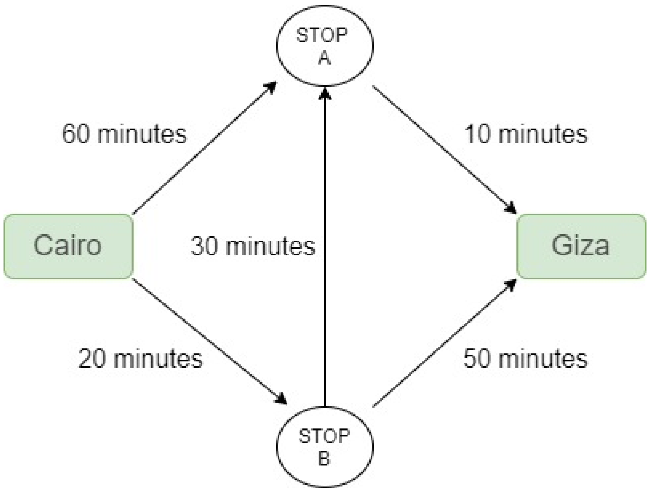


Graph Routes - shortest

- e.g. we can see time difference between purple and red paths
  - *from Cairo to Giza*
- red path is faster, in spite of one more segment
- to find the fastest path
  - *may use a different option to breadth-first search*
- we may use *Dijkstra's algorithm* to help us query graph for fastest path

graphs - Dijkstra's algorithm - working example 1 - part 1

- if we consider the following basic weighted graph
  - *may initially see how Dijkstra's algorithm works*
  - *i.e. to help us find the fastest path*



Dijkstra - example graph

- each segment in this graph has a corresponding cost
  - *time in minutes*
- may use *cost* with Dijkstra's algorithm
  - *calculate shortest possible time from Cairo to Giza...*

Algorithms and Data Structures

graphs - Dijkstra's algorithm - working example 1 - part 2

- may initially use the following steps with Dijkstra's algorithm
  - *i.e. to calculate fastest path from a defined start to finish in the graph*
  - *for the current start node*
- *1. identify the cheapest node*
  - *i.e. next node we can reach in least amount of time*
- simply check neighbour nodes for current node
  - *identify path with shortest time*
- e.g. from our starting point of *Cairo* there are two options
  - *either Stop A with a time of* `60 minutes`
  - *or Stop B with a time of* `20 minutes`
- we don't know values of other nodes at this point of search
- as we don't yet know how long it will take to get to finish
  - *Giza defined with an overall time of* `infinity`*...*

graphs - Dijkstra's algorithm - working example 1 - part 3

- we know closest node from start, *Cairo*
- *Stop B* with a time of `20 minutes`

node	time
Stop A	60 minutes
Stop B	20 minutes
Giza	infinity

graphs - Dijkstra's algorithm - working example 1 - part 4

- *2. update any costs for neighbours of this node*

- need to calculate all times from *Stop B* to available neighbour nodes
- in current example
  - *includes Stop A and our finish node of Giza*
- may now update our times from start, *Cairo*
  - *update to each node currently known in the graph*

node	time
Stop A	50 minutes
Stop B	20 minutes
Giza	70 minutes

Algorithms and Data Structures

graphs - Dijkstra's algorithm - working example 1 - part 5

- first improvement is a faster time from *Cairo* to *Stop A*
  - *even though we have to go through node Stop B*
- with current known neighbour nodes
  - *may also follow a path from start node, Cairo,*
  - *follow to finish in Giza*
  - *path takes* `70 minutes`
- currently have a shorter path from *Cairo* to *Stop A*
- plus a shorter path to finish from start...

graphs - Dijkstra's algorithm - working example 1 - part 6

- *3. repeat this pattern for each node in the graph*
- may now repeat pattern to check for other neighbour nodes
  - *potentially faster routes from start to finish...*
- repeat first step again
  - *need to find next node with shortest travel time*
- checked all of the neighbour nodes for *Stop B*
- we can now check next fastest neighbour of start node *Cairo*
- in current example
  - *this will be node Stop A...*

graphs - Dijkstra’s algorithm - working example 1 - part 7

- don’t need to update time from start node
  - *i.e. from Cairo to node Stop A*
  - *already identified a faster route...*
- we may check times for quickest route to finish
- now have an extra path to check
  - *i.e. from Stop A to finish in Giza*
- gives us a shorter time from start to finish
  - *due to its time of `10` minuutes...*

Algorithms and Data Structures

graphs - Dijkstra's algorithm - working example 1 - part 8

- now update our times as follows

node	time
Stop A	50 minutes
Stop B	20 minutes
Giza	60 minutes

- i.e. define fastest routes for following paths
  - *Cairo to Stop A = `50` minutes*
  - *Cairo to Stop B = `20` minutes*
  - *Cairo to Giza = `60` minutes*
- able to identify a quicker path from start to *Stop A*
  - *and a quicker path from start to finish...*

Algorithms and Data Structures

graphs - Dijkstra's algorithm - working example 1 - part 9

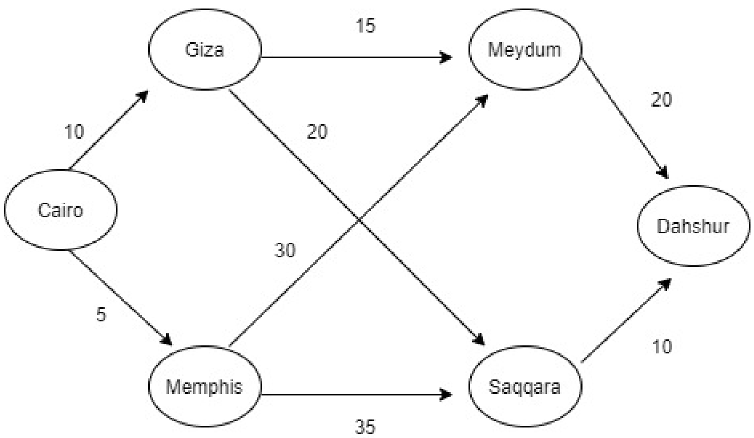
- calculate the time for the final path
  - *may currently define final path*
  - *calculated fastest time of* `60 minutes`*…*
- if we compare this calculation with a search using *breadth-first*
  - *may see that it would not have found that path*
- *breadth-first* would have found shorter path
  - *but a slower path in this example graph…*

graphs - Dijkstra's algorithm - working example 1 - benefits

- in current example, we may see an initial benefit
  - *benefit relative to this context*
  - *e.g. for a search with Dijkstra's algorithm compared with a breadth-first search*
- may use breadth-first search to find shortest path
  - *e.g. between defined nodes in graph*
- may use Dijkstra's algorithm to assign weights to graph
  - *use to find path with smallest total of calculated weights...*

graphs - Dijkstra's algorithm - working example 2 - part 1

- consider another working example for a graph with weighted edges
- e.g. a graph with values for cost to travel from one node to another
  - *perhaps from Cairo to Giza or Giza to Saqqara...*



Graph Weighted

- in this graph
  - *define weights for associated costs of travel along each edge*
- i.e. travel from *Memphis* to *Meydum* for `30`
  - *or, perhaps, from Giza to Meydum for only* `15`*...*

graphs - Dijkstra’s algorithm - working example 2 - part 2

- if we consider this graph
  - *may need to calculate cheapest route from Cairo*
  - *e.g. start point to an end point of Dahshur…*
- may use Dijkstra’s algorithm to perform this calculation
  - *follow defined four steps for this algorithm*
- initial costings may be defined as follows

parent	node	cost
Cairo	Giza	10
Cairo	Memphis	5
N/A	Meydum	infinity
N/A	Saqqara	infinity
N/A	Dahshur	infinity

- and set an initial parent for each node…

graphs - Dijkstra's algorithm - working example 2 - part 3

- then update this table as we execute the algorithm
- *1. start by finding cheapest node*
  - *in this graph, from a starting node of Cairo cheapest edge is `5` to Memphis*
  - *we can't make this initial path any cheaper*
  - *cheapest node will be Cairo to Memphis*
- *2. then, calculate cost to neighbours of this cheapest node, i.e. from Memphis*
  - *we now have costs for Meydum, `30`, and Saqqara, `35`*
  - *we can update our table of costs from our starting point to each neighbour*
    - Cairo to Meydum and Cairo to Saqqara
  - *Cairo -> Memphis -> Meydum*
  - *Cairo -> Memphis -> Saqqara*

graphs - Dijkstra’s algorithm - working example 2 - part 4

parent	node	cost
Cairo	Giza	10
Cairo	Memphis	5
Memphis	Meydum	35
Memphis	Saqqara	40
N/A	Dahshur	infinity

- now have costs for Meydum and Saqqara
- may define their costs as we travel through Memphis node
- as we can see in the table
  - *their parent node may also be updated to Memphis...*

graphs - Dijkstra's algorithm - working example 2 - part 5

- may now repeat these two steps for next cheapest node from Cairo
  - *i.e. Giza at a cost of `10`*
  - *update its values in the table as well*

parent	node	cost
Cairo	Giza	10
Cairo	Memphis	5
Giza	Meydum	25
Giza	Saqqara	30
N/A	Dahshur	infinity

- the costs for travel from starting point, Cairo, to Meydum and Saqqara updated
  - *they are cheaper now*
  - *so we update the values in the table*
- i.e. it's now cheaper to travel to Meydum and Saqqara via Giza…

graphs - Dijkstra's algorithm - working example 2 - part 6

- we may check cost to travel to end point, *Dahshur*
- check cheapest node from Giza
  - *currently Meydum at* `15`
- may update its neighbours
  - *gives us an initial cost for Dahshur of* `20`
- if we update table at this point
  - *we get the following travel cost from Cairo to Dahshur*

parent	node	cost
Cairo	Giza	10
Cairo	Memphis	5
Giza	Meydum	25
Giza	Saqqara	30
Meydum	Dahshur	45

- we finally have an initial travel cost from start to finish of `45`
  - *i.e. from Cairo -> Giza -> Meydum -> Dahshur*

Algorithms and Data Structures

graphs - Dijkstra’s algorithm - working example 2 - part 7

- we may also check next cheapest node from Giza, *Saqqara*
- then, we may travel from Saqqara to Dahshur
  - *for a total of `40` from Cairo*
- i.e. may now update cost of travel from start to finish
  - *update to a lower overall cost of `40`*

parent	node	cost
Cairo	Giza	10
Cairo	Memphis	5
Giza	Meydum	25
Giza	Saqqara	30
Saqqara	Dahshur	40

Algorithms and Data Structures

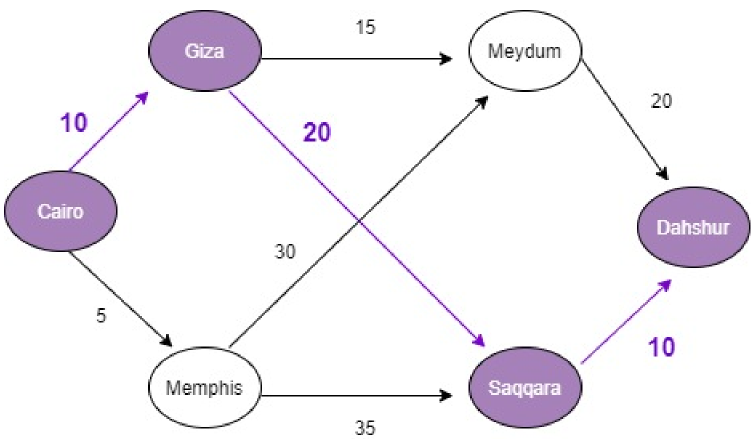
graphs - Dijkstra's algorithm - working example 2 - part 8

parent	node	cost
Cairo	Giza	10
Cairo	Memphis	5
Giza	Meydum	25
Giza	Saqqara	30
Saqqara	Dahshur	40

- we may see that shortest path costs `40`
- using this overall cost
  - *now define path for travel from start to finish in this graph*
- to help with this path definition
  - *may check parent node set in last table*
- i.e. ended up with *Saqqara* as parent for end point *Dahshur*

graphs - Dijkstra’s algorithm - working example 2 - part 9

- we know that we need to travel from Saqqara to Dahshur
- may then follow path to parent of Saqqara, set to Giza
- i.e. to travel to Saqqara we need to begin at Giza
- we follow Giza back to its parent
  - *starting point at Cairo*
- now have a complete route for traveling from start point to end point in least cost, `40`



Graph Weighted - Final Costs

Resources

various

- Dijkstra’s algorithm
- Graph - abstract data type

videos

- Dijkstra’s algorithm