COMP20003 Workshop Week 10

1 Graphs: concepts
2 Graph representation
3 BFS & Dijkstra’s Algorithm
4 Teaching Surveys
5 Lab: Simple Implementation of Dijkstra

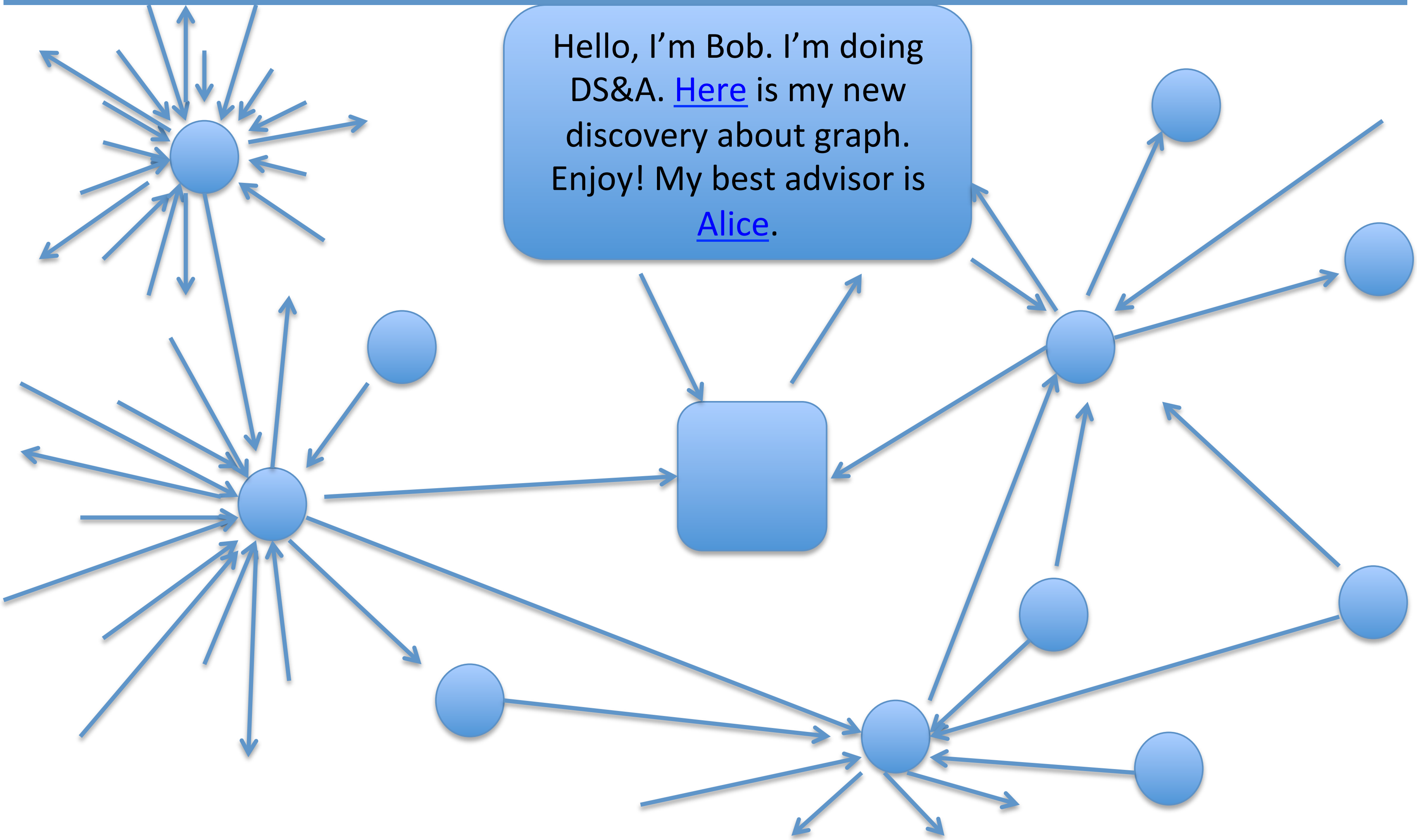
Graphs

Visualization

Any problem where graph can be applied?

Are link lists and trees graphs? What special about them?

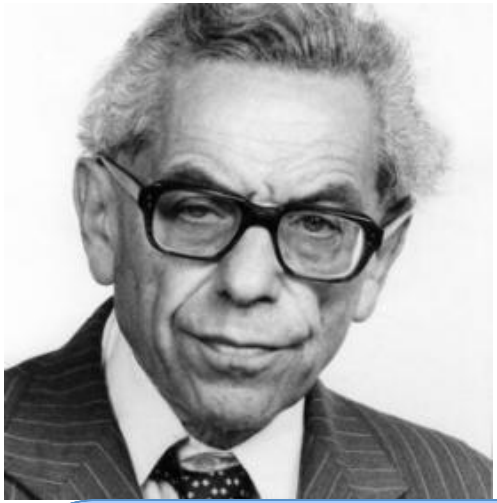
PageRank: Bob becoming famous!



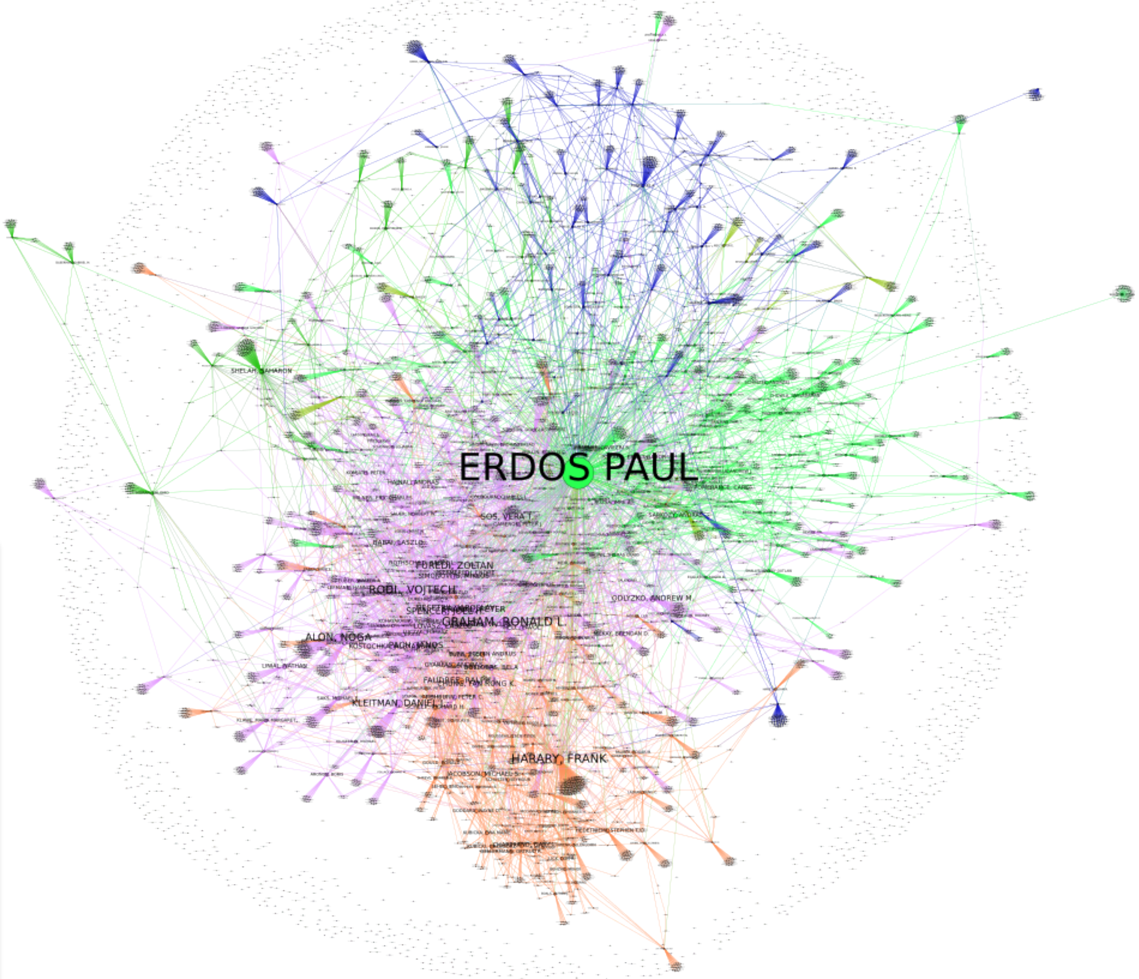
Hello, I'm Bob. I'm doing DS&A. Here is my new discovery about graph. Enjoy! My best advisor is Alice.

Erdős' Collaborators Graph (6927 nodes)

(http://genedan.com/tag/erdos-network-graph/)



Erdos (1913-1996) is a great math-cian. He wrote ~1500 papers, co-authored with >500 authors…



Graphs: Concepts

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

$V=\{v_i\}$: set of *vertices*, or *nodes*

$E=\{(v_i, v_j) \mid v_i \in V, v_j \in V\}$: set of *edges*, *arcs*, or *links*;

$|V|$ is called the *order* of the graph

$|E|$ is called the *size* of the graph

dense and *sparse* graphs

directed, *di-graph*, *undirected*, *acyclic*, *DAG*

connected graph, connected component

weakly and strongly connected components

Graphs: Representation

Representation (general approaches):

1. *Adjacency matrix* (what is that?)
2. Set of *adjacency lists* (what is that?)

What is a suitable representation method for:

- a graph of this class, where nodes represent students, edge (a,b) means “a knows b”,
- the Erdős’s collaboration graph (check your answer by visiting the website),
- the webgraph?

Graph: Adj matrix or list?

Space:

- Adjacency matrix: require m^2 bits, where $m = |V|$
- Set of adjacency lists: requires

n * sizeof(list_node) + m * sizeof(pointer), where $n = |E|$

Speed:

- Adjacency matrix is faster (in general) because of the random access

In practice: graphs are often sparse, or at least not dense enough, and so adjacency lists representation is often preferred. Choosing A/Matrix or A/List is a matter of trade-off between speed and space

Graphs

Q 9.1

Construct weighted graphs from records whose format is *vertex, vertex′, weight*. For example, in a weighted, directed graph, the record 1, 3, 500 means that there is an edge from vertex 1 to vertex 3 with weight 500.

The following questions all refer to the input data below

```
1 2 200
1 3 100
1 4 500
2 3 150
2 4 300
4 5 100
```

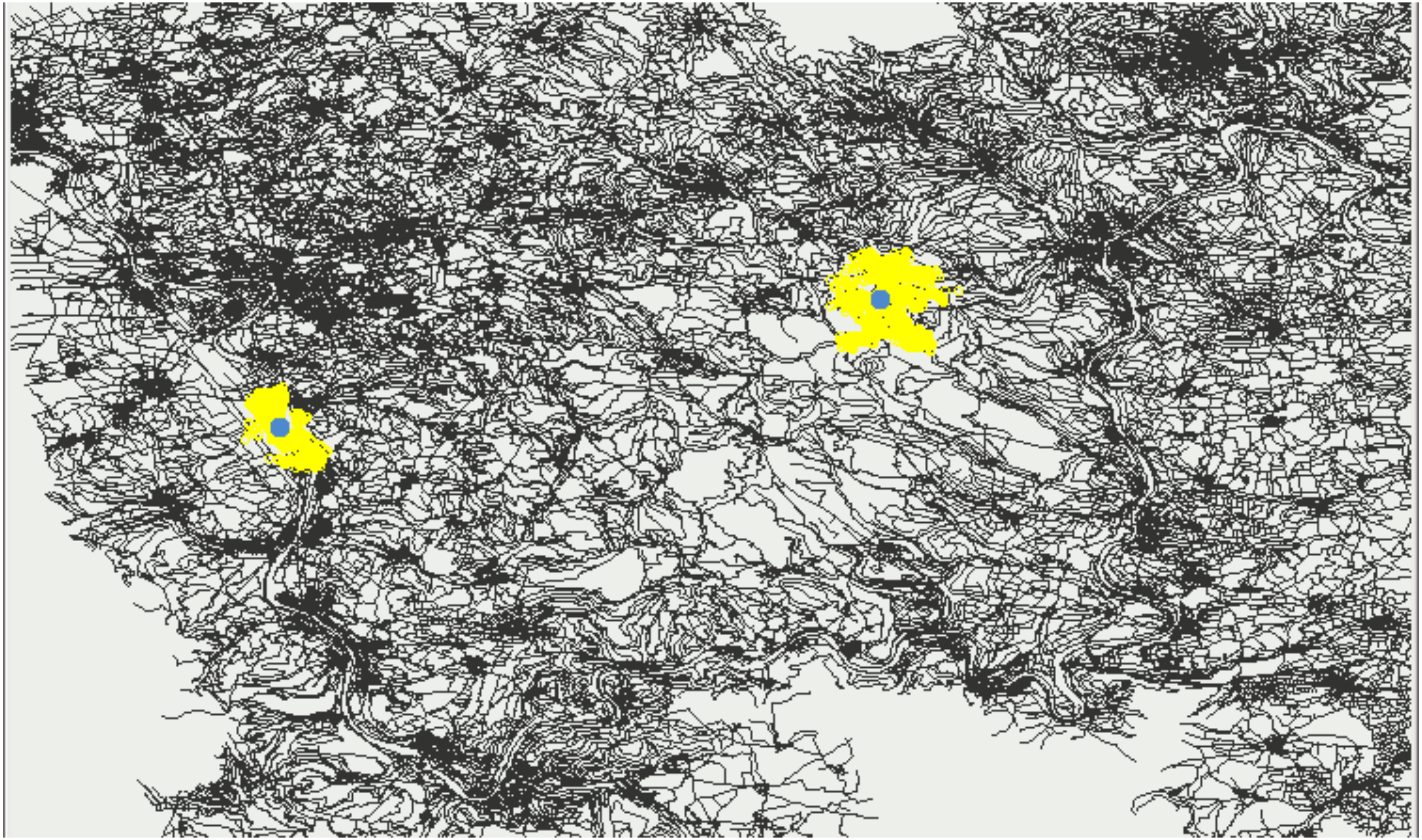
Q 9.1

Data

1 2 200
1 3 100
1 4 500
2 3 150
2 4 300
4 5 100

1. **Draw a weighted directed graph that reflects these edges and weights (logical representation).**
2. **Construct an adjacency matrix for the weighted digraph you have just drawn, including the weights. Be explicit about how you are going to handle matrix cells for which there is no information in the data, and self-loops (edges between a node and itself)**
3. **If you are told that these inputs refer to a weighted undirected graph, how would your adjacency matrix change?**
4. **Which representation, adjacency matrix or adjacency list, would be most suitable for this graph?**

Breath-first Search (BFS): visit all neighbors first



DFS from a single vertex

The task:

- Given a weighted graph `G=(V,E)`, and `s∈V`
- Visit all vertices which are reachable from `s`.

The algorithm:

```
enqueue(s, Q)
while (Q is not empty) {
    x= dequeue(Q)
    visit x
    for all v that (x, v) ∈ E: enqueue(v, Q)
}
```

Dijkstra’s Algorithm

The task:

- Given a weighted graph `G=(V,E,w(E))`, and `s∈V`, and supposing that *all weights are positive*.
- Find shortest path (path with min total weight) from `s` to all other vertices.

QoCT: do it now

QoCT surveys will be open on 2 October 2017 (Week 10).

The link is at:

`https://apps.eng.unimelb.edu.au/casmas/index.php?r=qoct/subjects`

Dijkstra’s Algorithm as a (special) BFS

Basic idea

if A → B → C → D is a shortest path

then

A → B → C is a shortest path (from A to C)

A → B is a shortest path (from A to B)

Dijkstra’s Algorithm as a (special) BFS

Basic idea

if `s` → `A` → `B` is a shortest path then `s` → `A` is a shortest path.

Init:

- start with `dist[s]= 0`, and `dist[*]`= , set `unvisited_set= V`

Round 1:

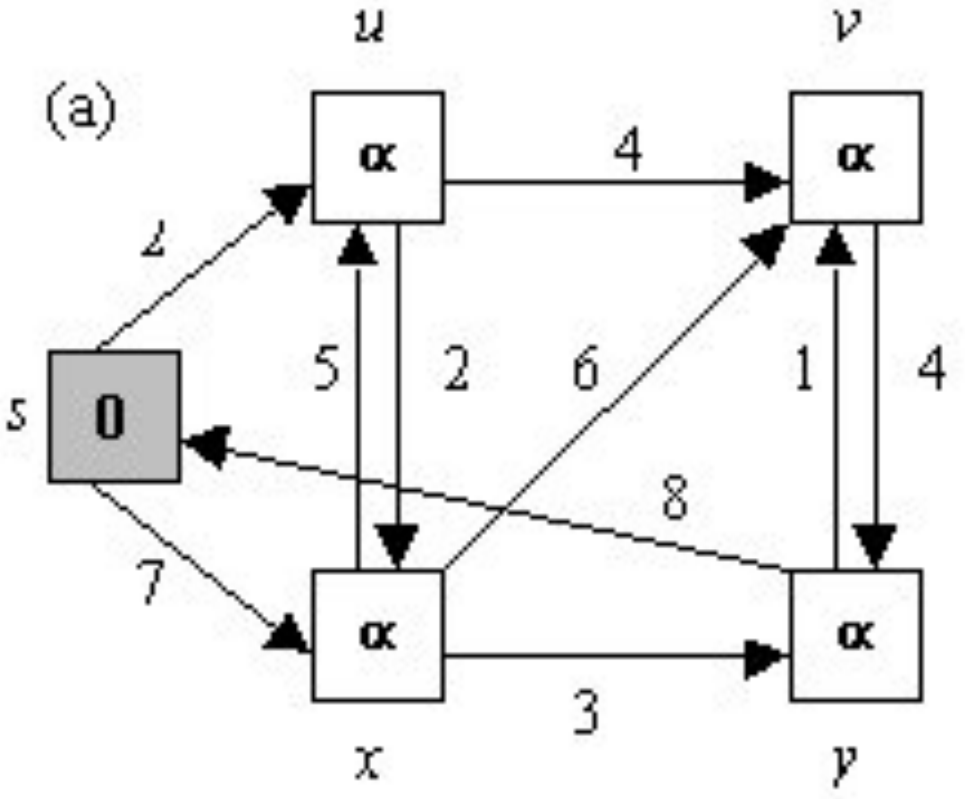
- choose node with min `dist[],` which is `s`;
- visit all nodes u adjacent to s and update `dist[u]`;
- mark s as visited (remove it from the `unvisited_set)`;

Round 2:

- choose the node with min dist[] from `unvisited_set`
- do the other steps as in Round 1

Dijkstra algorithm from vertex s: initialization

In each round, one element will be removed from A and added to B. Repeat round until A become empty.



B= { }

A= V = { s, u , v , x , y }

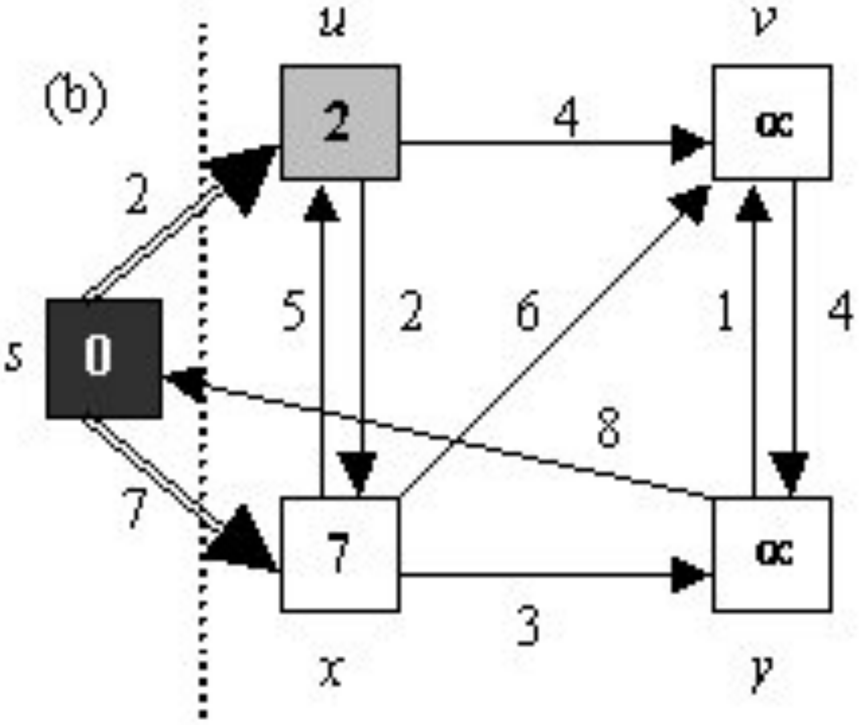
dist = { **0** , ∞ , ∞ , ∞ , ∞ }

prev = { nil, nil , nil , nil , nil }

Dijkstra algorithm from vertex s

Round 1:

- choose s (node with least dist[] in A)
- update dist[] of nodes adjacent to s *if applicable*



B = { s }

dist= { 0 }

A = { , u , v , x , y }

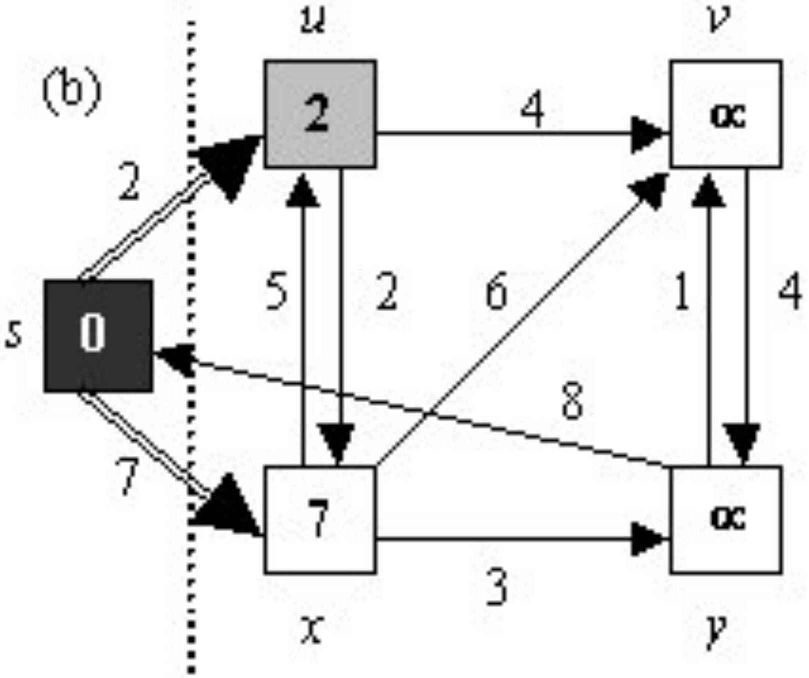
dist = { , **2** , ∞ , 7 , ∞ }

prev = { nil, s , , s , }

Dijkstra algorithm from vertex s

Round 1:

	s	u	v	x	y
0	0 N	∞ N	∞ N	∞ N	∞ N
1		2 s	∞ N	7 s	∞ N



B = { s }

dist= { 0 }

A = { , u , v , x , y }

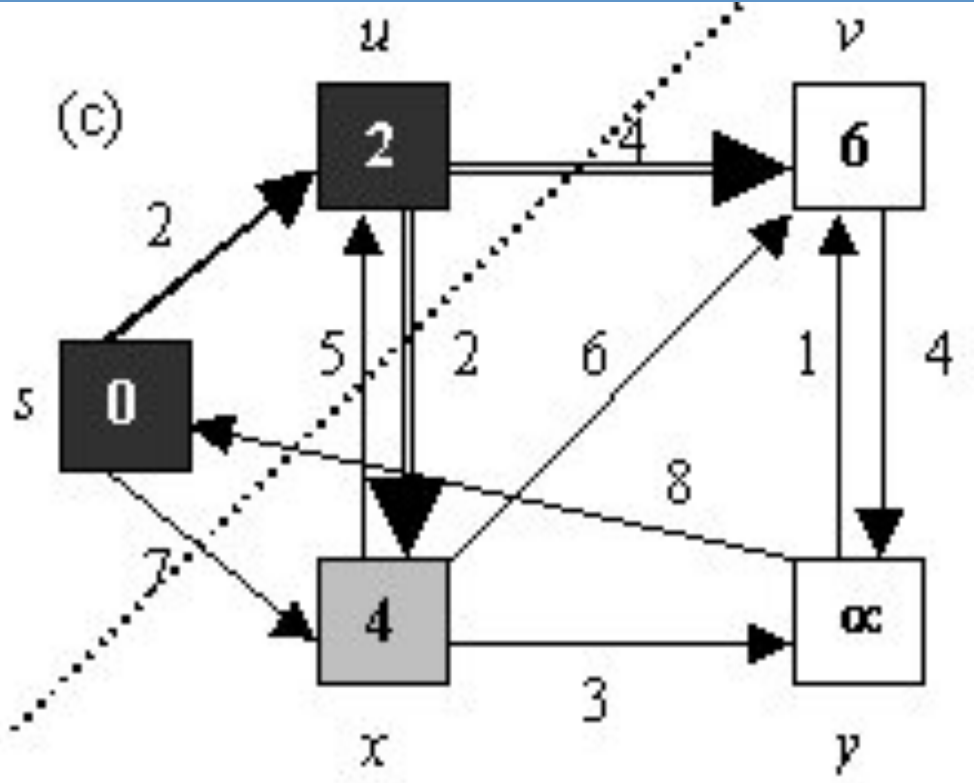
dist = { , **2** , ∞ , 7 , ∞ }

prev = { nil, s , , s , }

Dijkstra algorithm from vertex s

Round 2:

- choose u (node with least dist[] in A)
- update dist[] of nodes adjacent to s *if applicable*

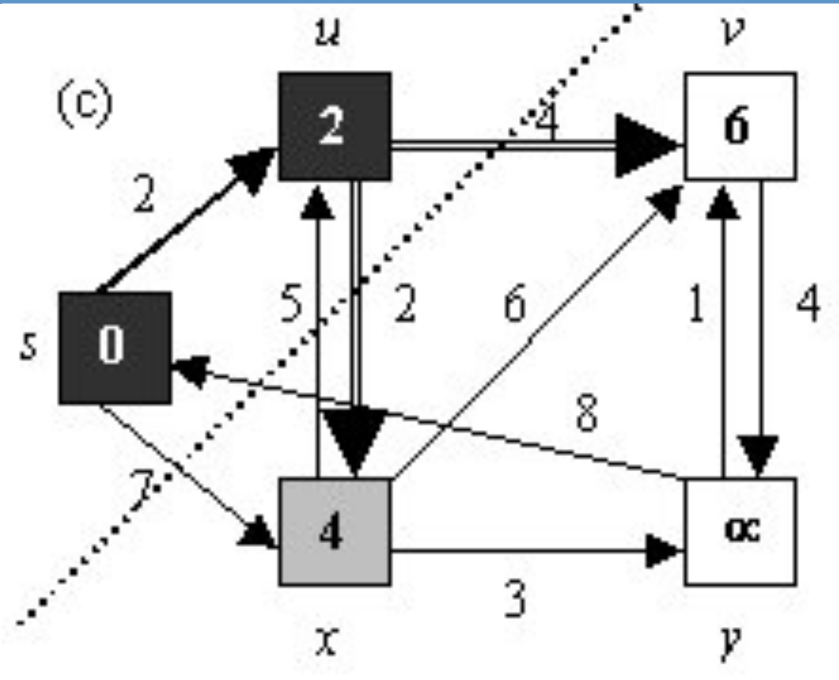


B= { s, u}
{0, 2}
{ nil, s }

A = { , , v , x , y }
dist = { , , 6 , 7→**4** , ∞ }
prev = { N, s , u, u , N }

Dijkstra algorithm from vertex s

	s	u	v	x	y
0	0 N	∞ N	∞ N	∞ N	∞ N
1		2 s	∞ N	7 s	∞ N
2			6 u	**4** u	∞ N



B= { s, u}

{0, 2}

{ nil, s }

A = { , , v , x , y }

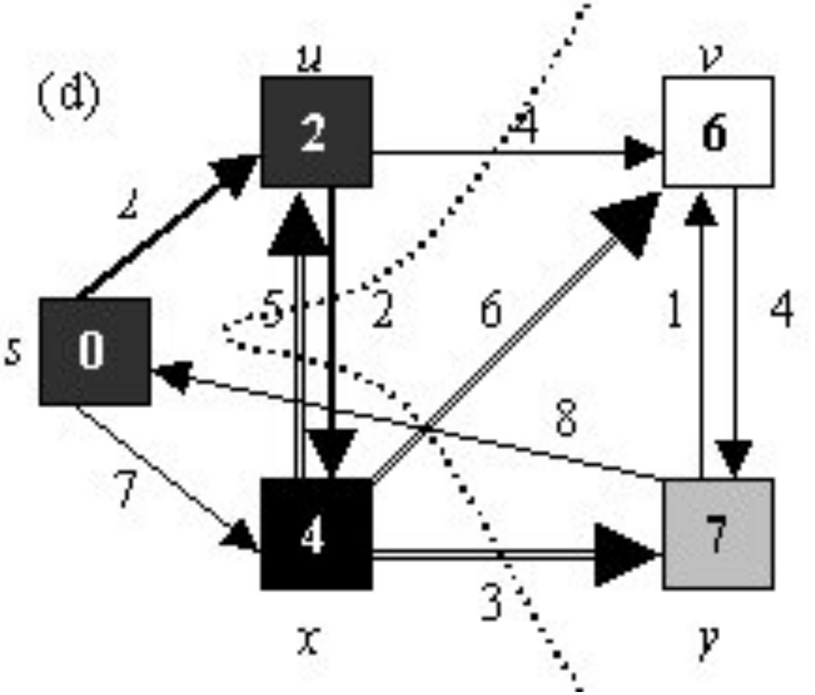
dist = { , , 6 , 7→**4** , ∞ }

prev = { N, s , u, u , N }

Dijkstra algorithm from vertex s

Round 3

	s	u	v	x	y
0	0 N	∞ N	∞ N	∞ N	∞ N
1		2 s	∞ N	7 s	∞ N
2			6 u	**4** u	∞ N
3			**6** u		7 x



B= { s, u, x }

{0, 2, 4 }

{ nil, s, u }

A = { , , v , , y }

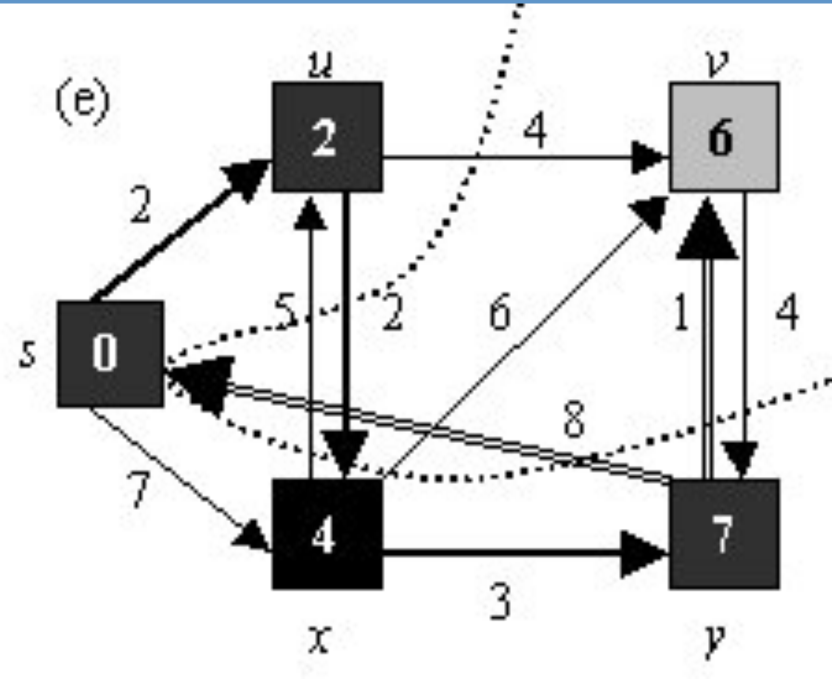
dist = { , , **6** , , 7 }

prev = { , , u , , x }

Dijkstra algorithm from vertex s

Round 4

	s	u	v	x	y
0	0 N	∞ N	∞ N	∞ N	∞ N
1		2 s	∞ N	7 s	∞ N
2			6 u	**4** u	∞ N
3			**6** u		7 x
4					**7** x



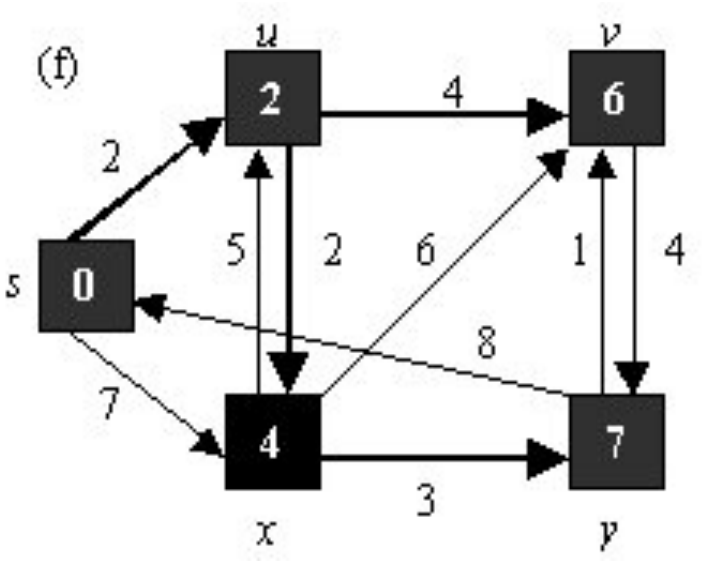
B= { s, u, x, v}

A = { y }

Dijkstra algorithm from vertex s

Round 5

	s	u	v	x	y
0	0 N	∞ N	∞ N	∞ N	∞ N
1		2 s	∞ N	7 s	∞ N
2			6 u	**4** u	∞ N
3			**6** u		7 x
4					7 x
5					



B= { s, u, x, v, y} A = { }

Question: what is the shortest path from s to y?

Dijkstra's algorithm [conceptual only]

Purpose: Find shortest path from vertex s

```
set dist[u] = ∞, pred[u]=nil  for all u, then dist[s]= 0;
set B= ∅, A= V.   B is set of vertices where shorted path from s found, A is set of other vertices.
while (A is not empty):
   select u from A such that dist[u] is smallest
   remove u from A and add it to B
   for all (u,v) in G:
      if (dist[v] > dist[u]+w(u,v)): update dist[v] and pred[v]
```

Q: How to represent A?

Dijkstra's algorithm [conceptual only]

```
set dist[u]= ∞, pred[u]=nil for all u, then dist[s]= 0;
set B= ∅, A= makePQ(V)
while (A not empty)
   u= deleteMin(A)
   add u to B
   for all (u,v) in G:
      if (dist[v] > dist[u]+w(u,v): update dist[v]
      and pred[v]
```

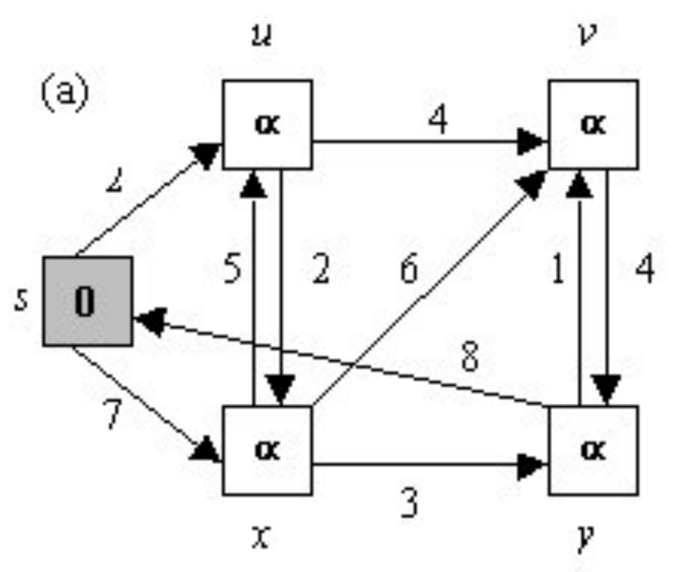
“update dist[v] and pred[v]” means

dist[v]= dist[u]+w(u,v) ;   pred[v]= u

decrease weight of v in PQ to dist[v], hence, need to locate v in PQ, change weight and sift-up

Lab

Data	Task

5 s u v x y
0 1 2
0 3 7
1 2 4
1 3 2
…
 u
v
(a)
α
α
4
2
5
2
6
1
4
s
0
8
7
α
α
3
x
y
	Write typedef for the graph, using adjacent lists:
	- supposing MAX_N = 100
	- define data type vertex_t
	- define vertices as array of vertex_t
	- define the adjacent lists properly
	
	Implement simple Dijkstra algorithm:
	
	`void dijkstra(graph_t *G,`
	`        int dist[], int prev[])`
	
	- just use an array for the (priority) queue, ie doing linear search in the array when looking for element with min dist[]

Assignment 2: Minimal Work to do

to fill in code within:

- `search.c` (search for FILL IN CODE): finish programming Dijkstra, and
- `extensions.c` (search for FILL IN CODE): program the dead-end detection

type ./flow -h to see options

Assignment 2: Work to do

- Explore the system, starting from `flow_solver.c`;
- Understanding the definitions, declarations, especially:
  - representation of the board,
  - information on the state of the board, and
  - all that are directly manipulated in `dijkstra()`;
- Understanding the overall mechanism of the whole system; and of course
- Modify the 2 functions (possibly with adding some helping functions for them).

NOTE: you have to use the supplied structure of `dijkstra`, and don't try to replace it by something else (for example, some "brilliant" or "easy to understand" things you might find in the Internet).