Voorbereiding Programmeerwedstrijden

najaar 2021

`https://liacs.leidenuniv.nl/~vlietrvan1/vbpw/`

Rudy van Vliet
kamer 140 Snellius, tel. 071-527 2876
rvvliet(at)liacs(dot)nl

college 5, 11 oktober 2021

Graph Traversal

Graph Algorithms

(Eind)Programmeerwedstrijd

maandag 1 november, 09.15-13.15 uur

8.6.3. Queue

Possible reasons not to extend current partial permutation:

- too many persons that are or will become visible from the left
- too few persons that are or may become visible from the left
- too many persons that are or will become visible from the right
- too few persons that are or may become visible from the right
- too few persons that may become visible from either left or right together

In all these cases, return 0

Useful to distinghuish between situations that tallest person has / has not yet been placed into permutation.

9. Graph Traversal

graph is unifying theme of computer science

9.2. Data Structures for Graphs

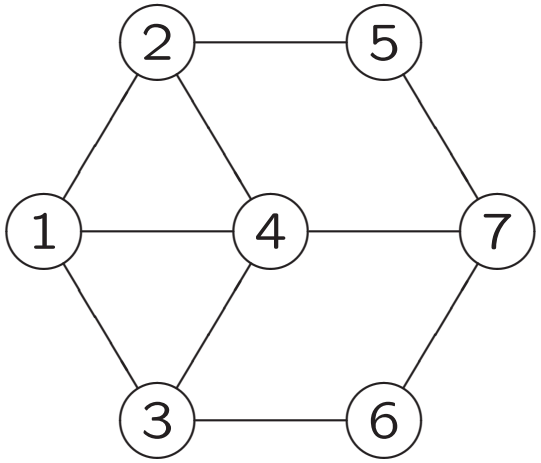
- adjacency matrix
  - \+ quick access, insertion, deletion of edge
  - − much space (e.g., Manhattan), slow iteration over neighbours

- adjacency list in lists
  - − slow access, deletion of edge
  - \+ less space, quick iteration over neighbours
    (LKP2021. Candy Contribution)

- adjacency list in matrix

- table of edges (for Kruskal)

Adjacency List in Matrix



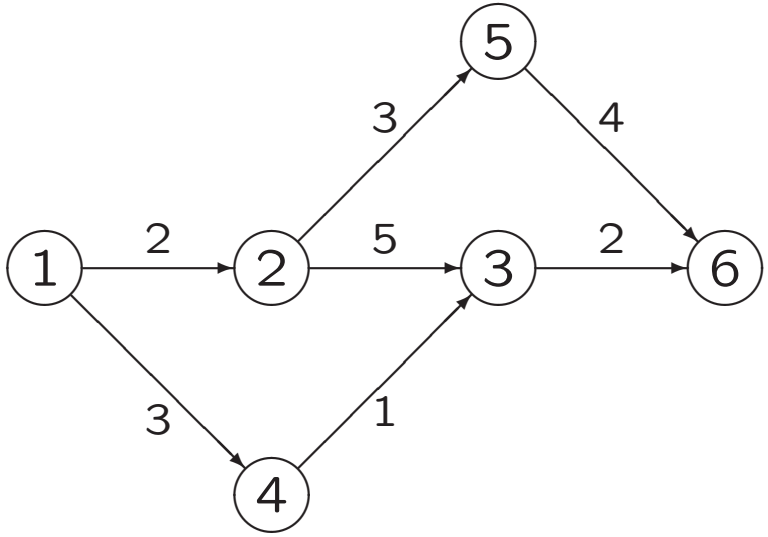
node	degree	neighbours			
		0	1	2	3
1	3	2	3	4	
2	3	1	4	5	
3	3	1	4	6	
4	4	1	2	3	7
5	2	2	7		
6	2	3	7		
7	3	4	5	6	

- quick iteration over neighbours
- no pointers

9.5. Topological Sorting

- of DAG
- all directed edges from left to right
- useful
  - for schedule respecting precedence constraints
  - for finding shortest / longest path from x to y (with dynamic programming)...
- implementation
  - variant of dfs
  - * count incoming edges per vertex
    * maintain queue of vertices without incoming edges
- see Algoritmiek

Topological Sorting (Example)



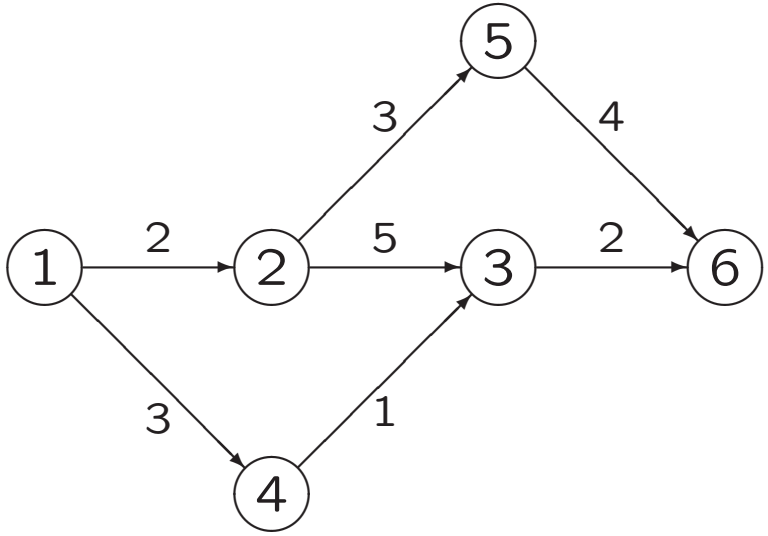
finding longest path in DAG. . .

9.6.5. Edit Step Ladders

9.6.6. Tower of Cubes

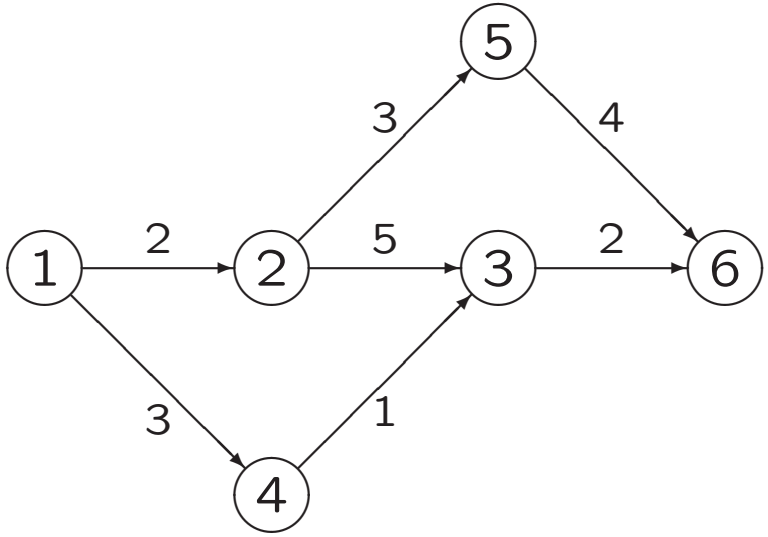
9.6.8. Hanoi Tower Troubles Again!

10.4. Network Flows and Bipartite Matching



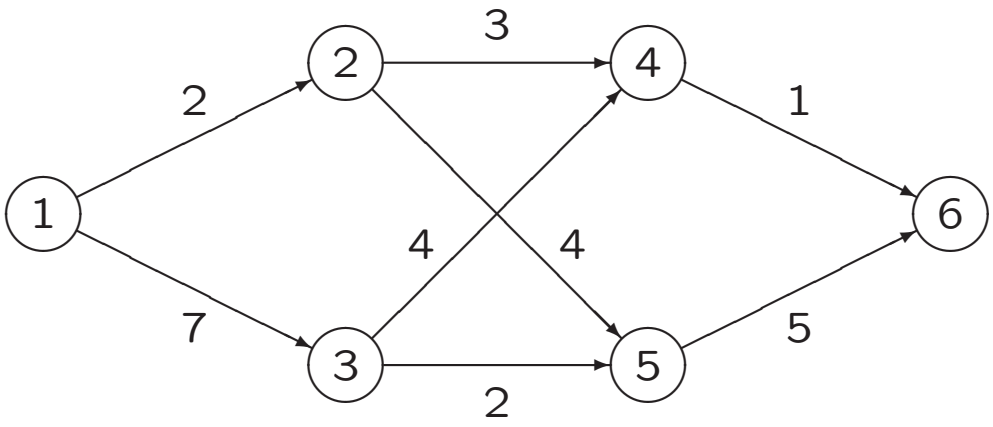
maximum flow from 1 to 6 is . . .

Ford-Fulkerson Algorithm (Example)



repeat finding augmenting paths

Ford-Fulkerson Algorithm (Example)

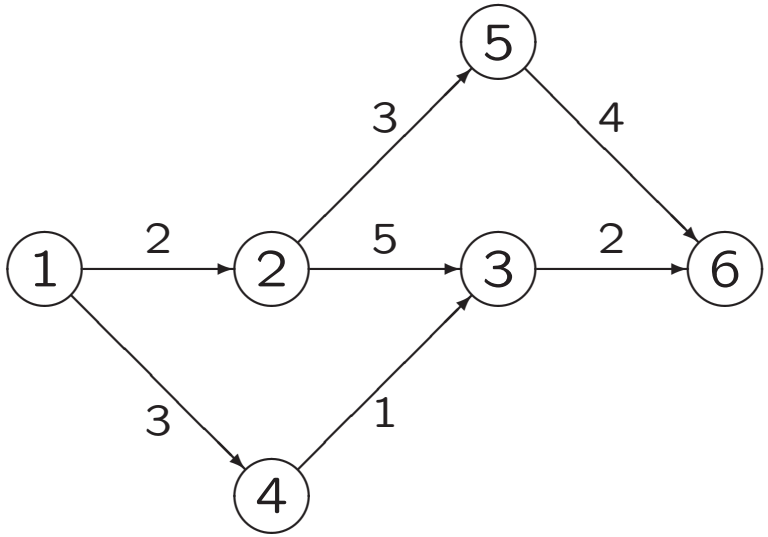


maximum flow from 1 to 6 is . . .

Ford-Fulkerson Algorithm

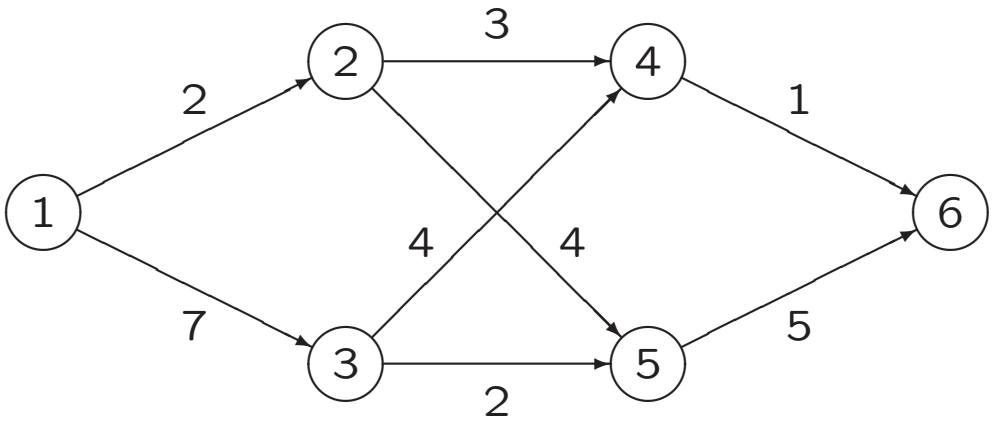
```
void netflow (flowgraph *g, int source, int sink)
{ int volume;    // weight of augmenting path

  add_residual_edges (g);

  dfs (g, source);
  volume = path_volume (g, source, sink, parent);

  while (volume>0)
  { augment_path (g, source, sink, parent, volume);
    dfs (g, source);
    volume = path_volume (g, source, sink, parent);
  }
}
```

Minimum Cut



minimum cut of 1 and 6 is . . .

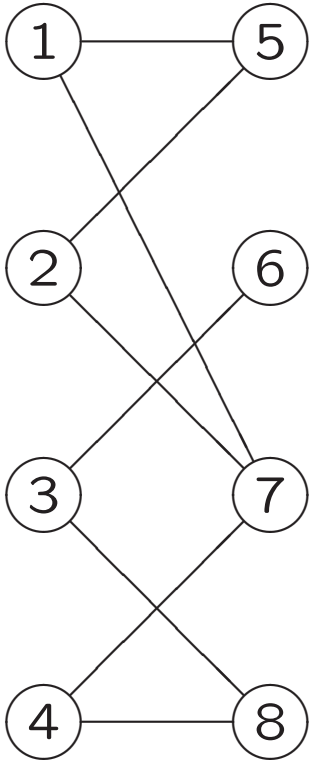
Minimum Cut



- minimum cut of 1 and 6 is . . .

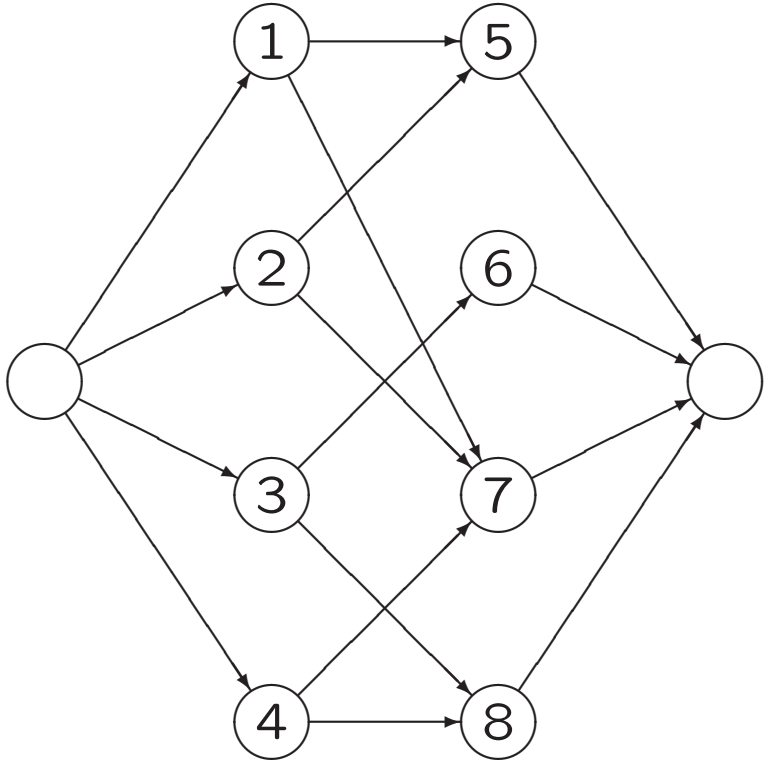
- finding minimum cut. . .

Maximum Bipartite Matching



maximum matching. . .

Maximum Bipartite Matching



maximum matching $\approx$ maximum flow from source to sink

Evacuation

9.6.8. Hanoi Tower Troubles Again!

10.3. Shortest Paths

- Dijkstra's algorithm

// invoer: samenhangende gewogen graaf $G = (V, E)$ en startknoop s
// uitvoer: array dat de lengtes van de kortste paden vanuit s bevat;
// na afloop is pad[v] = de lengte van een kortste pad van s naar v

for $v \in V$ **do**
    pad[v] := ∞;
od
pad[s] := 0;
$U := \emptyset$;

while ($U \neq V$) **do**
    vind knoop $v^* \in V \setminus U$ met pad[v^*] minimaal;
    $U := U \cup \{v^*\}$;
    for alle knopen v aangrenzend aan v^* **do**
        if pad[v^*] + gewicht(v^*, v) < pad[v] **then**
            pad[v] := pad[v^*] + gewicht(v^*, v);
        fi
    od
od

Complexiteit (met adjacency matrix):
$\Theta(n + 1 + n(n + 1 + n)) = \Theta(n^2)$

// invoer: samenhangende gewogen graaf $G = (V, E)$ en startknoop s
// uitvoer: array dat de lengtes van de kortste paden vanuit s bevat;
// na afloop is pad[v] = de lengte van een kortste pad van s naar v

for $v \in V$ **do**
    pad[v] := ∞;
od
pad[s] := 0;
$U := \emptyset$;

while ($U \neq V$) **do**
    vind knoop $v^* \in V \setminus U$ met pad[v^*] minimaal;
    $U := U \cup \{v^*\}$;
    for alle knopen v aangrenzend aan v^* **do**
        if pad[v^*] + gewicht(v^*, v) < pad[v] **then**
            pad[v] := pad[v^*] + gewicht(v^*, v);
        fi
    od
od

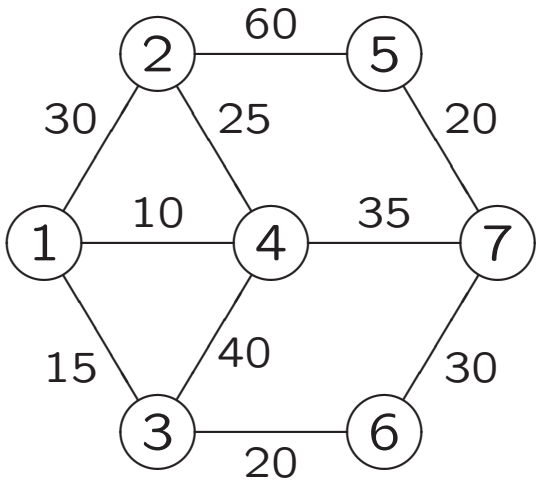
Complexiteit (met adjacency list en heap):
$O(n + 1 + n + n \log n + m \log n) = O(m \log n)$

LKP2021. Candy Contribution

10.2. Minimum Spanning Trees

- Prim's algorithm
- Kruskal's algorithm
- see Algoritmiek

Prim's Algorithm (Example)



9.6.3. The Tourist Guide