Sökning del 1

Tillämpad Algoritmisk Problemlösning (Våren 2026)

Martin Berglund, mbe@cs.umu.se

Låt oss lösa ett problem

Your job is to arrange n rambunctious children in a straight line, facing front. You are given a list of m statements of the form "i hates j". If i hates j, then you do not want put i somewhere behind j, because then i is capable of throwing something at j.

1. Give an algorithm that orders the line, (or says that it is not possible) in $O(m+n)$ time.

Låt oss lösa ett problem (2)

2. Suppose instead you want to arrange the children in rows, such that if i hates j then i must be in a lower numbered row than j. Give an efficient algorithm to find the minimum number of rows needed, if it is possible.

Grafer i många former

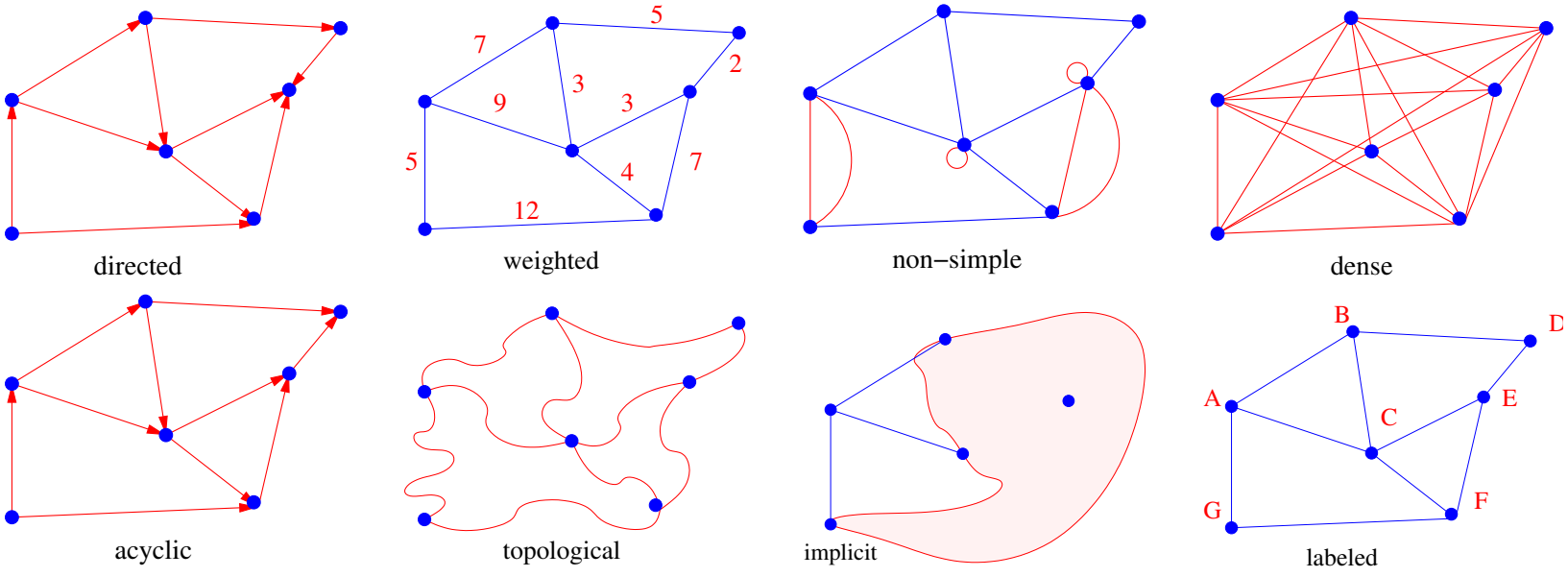


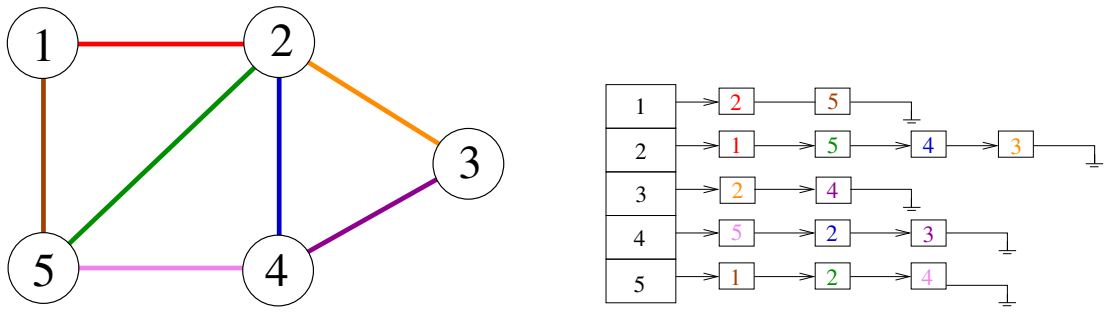
Figure 7.2: Important properties / flavors of graphs.

There are two main data structures used to represent graphs. We assume the graph $G = (V, E)$ contains n vertices and m edges.

We can represent G using an $n \times n$ matrix M, where element $M[i, j]$ is, say, 1, if (i, j) is an edge of G, and 0 if it isn't. It may use excessive space for graphs with many vertices and relatively few edges, however.

Can we save space if (1) the graph is undirected? (2) if the graph is sparse?

An *adjacency list* consists of a $N \times 1$ array of pointers, where the ith element points to a linked list of the edges incident on vertex i.



To test if edge (i, j) is in the graph, we search the ith list for j, which takes $O(d_i)$, where d_i is the degree of the ith vertex. Note that d_i can be much less than n when the graph is sparse. If necessary, the two *copies* of each edge can be linked by a pointer to facilitate deletions.

Tradeoffs Between Adjacency Lists and Adjacency Matrices

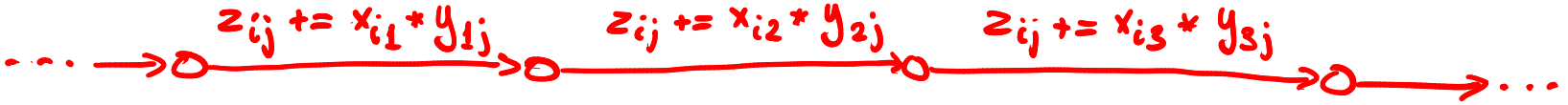
Comparison	Winner
Faster to test if (x, y) exists?	matrices
Faster to find vertex degree?	lists
Less memory on small graphs?	lists $(m + n)$ vs. (n^2)
Less memory on big graphs?	matrices (small win)
Edge insertion or deletion?	matrices $O(1)$
Faster to traverse the graph?	lists $m + n$ vs. n^2
Better for most problems?	lists

Both representations are very useful and have different properties, although adjacency lists are probably better for most problems.

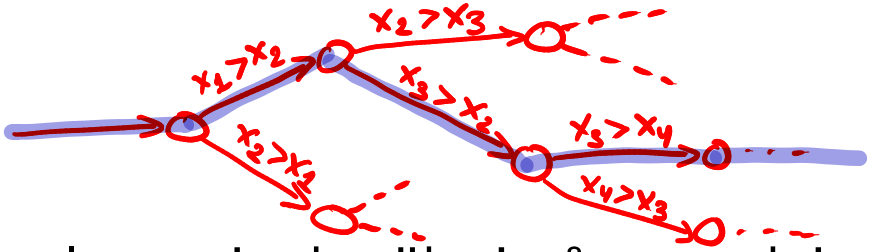
Sökning är fundamentalt

Sökning: mer *beräkningsmodell* än algoritmkategori, jämför

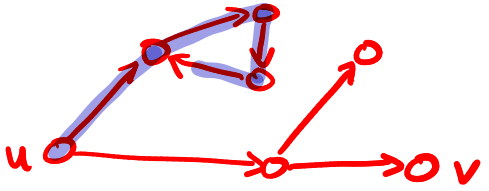
- Raka-linjen-beräkningar: bara *bara ett alternativ varje steg*
  - Matris-multiplikation, lång MADD-serie



- Algoritmer med en *förutbestämd "väg"*
  - Sortering: Upprepade jämförelser, $x_i < x_j$, resultatet avgör vilka jämförelser som görs härnäst (men "ändrar sig" aldrig)



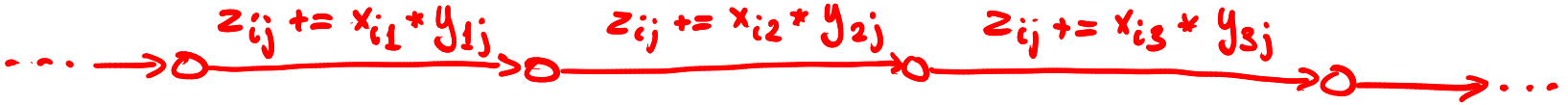
- Sökning: flera alternativ, besöks i någon ordning
  - Är $v \in V$ nåbar från $u \in V$ i grafen $G = (V, E)$?
  - Testa kanter, också sådana som visar sig inte leda till lösning



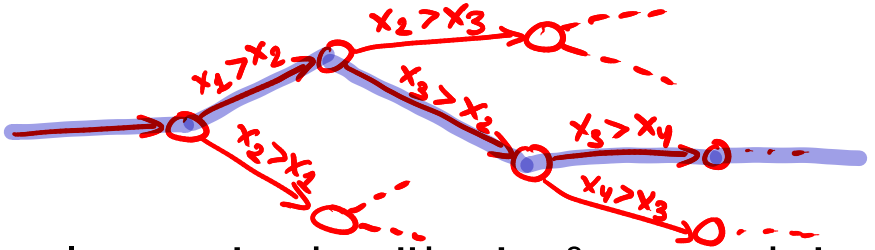
Sökning är fundamentalt

Sökning: mer *beräkningsmodell* än algoritmkategori, jämför

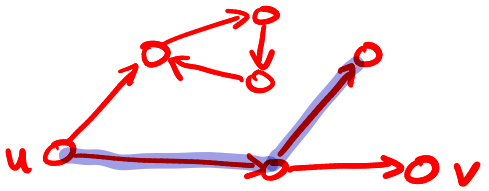
- Raka-linjen-beräkningar: bara *bara ett alternativ varje steg*
  - Matris-multiplikation, lång MADD-serie



- Algoritmer med en *förutbestämd "väg"*
  - Sortering: Upprepade jämförelser, $x_i < x_j$, resultatet avgör vilka jämförelser som görs härnäst (men "ändrar sig" aldrig)



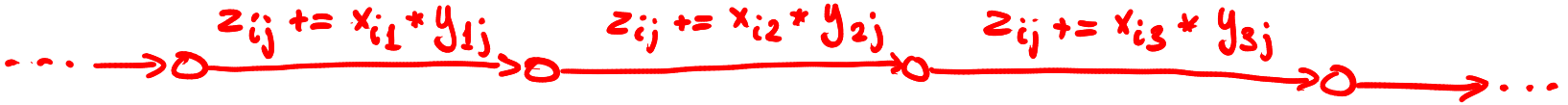
- Sökning: flera alternativ, besöks i någon ordning
  - Är $v \in V$ nåbar från $u \in V$ i grafen $G = (V, E)$?
  - Testa kanter, också sådana som visar sig inte leda till lösning



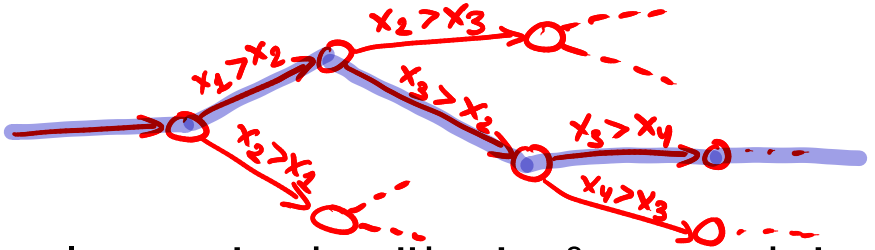
Sökning är fundamentalt

Sökning: mer *beräkningsmodell* än algoritmkategori, jämför

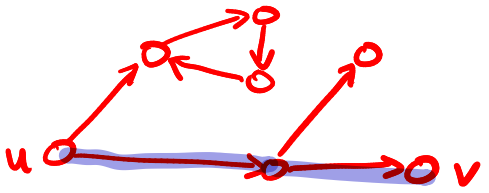
- Raka-linjen-beräkningar: bara *bara ett alternativ varje steg*
  - Matris-multiplikation, lång MADD-serie



- Algoritmer med en *förutbestämd "väg"*
  - Sortering: Upprepade jämförelser, $x_i < x_j$, resultatet avgör vilka jämförelser som görs härnäst (men "ändrar sig" aldrig)



- Sökning: flera alternativ, besöks i någon ordning
  - Är $v \in V$ nåbar från $u \in V$ i grafen $G = (V, E)$?
  - Testa kanter, också sådana som visar sig inte leda till lösning



Sökning är fundamentalt

Explicit graf G mer undantag än regel

- All *ickedeterminism:* realiseras deterministiskt med sökning
- Logik-programmering: Prolog, Kanren/core.logic, SMT, etc.
- All *brute-force:* testa "alla" alternativ, maskerad sökning

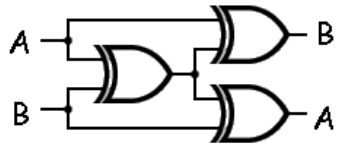
Sökning är fundamentalt

Explicit graf *G* mer undantag än regel

- All *ickedeterminism:* realiseras deterministiskt med sökning
- Logik-programmering: Prolog, Kanren/core.logic, SMT, etc.
- All *brute-force:* testa "alla" alternativ, maskerad sökning

Yet Satisfiability Again!

Alice recently started to work for a hardware design company and as a part of her job, she needs to identify defects in fabricated integrated circuits. An approach for identifying these defects boils down to solving a satisfiability instance. She needs your help to write a program to do this task.



Picture from Wikimedia Commons

Input

The first line of input contains a single integer, not more than 5, indicating the number of test cases to follow. The first line of each test case contains two integers n and m where $1 \leq n \leq 20$ indicates the number of variables and $1 \leq m \leq 100$ indicates the number of clauses. Then, m lines follow corresponding to each clause. Each clause is a disjunction of literals in the form Xi or ~Xi for some $1 \leq i \leq n$, where ~Xi indicates the negation of the literal Xi. The "or" operator is denoted by a 'v' character and is seperated from literals with a single space.

Output

For each test case, display `satisfiable` on a single line if there is a satisfiable assignment; otherwise display `unsatisfiable`.

Sample Input 1

```
2
3 3
X1 v X2
~X1
```

Sample Output 1

```
satisfiable
unsatisfiable
```

Submit | Stats

Problem ID: satisfiability

CPU Time limit: 3 seconds

Memory limit: 1024 MB

Difficulty: 3.9

Download:
Sample data files

Author: Babak Behsaz

Source: Rocky Mountain Regional Contest (RMRC) 2014

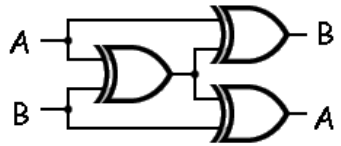
License:

Sökning är fundamentalt

Explicit graf *G* mer undantag än regel

- All *ickedeterminism:* realiseras deterministiskt med sökning
- Logik-programmering: Prolog, Kanren/core.logic, SMT, etc.
- All *brute-force:* testa "alla" alternativ, maskerad sökning

Alice recently started to work for a hardware design company and as a part of her job, she needs to identify defects in fabricated integrated circuits. An approach for identifying these defects boils down to solving a satisfiability instance. She needs your help to write a program to do this task.



Picture from Wikimedia Commons

Input

The first line of input contains a single integer, not more than 5, indicating the number of test cases to follow. The first line of each test case contains two integers n and m where $1 \leq n \leq 20$ indicates the number of variables and $1 \leq m \leq 100$ indicates the number of clauses. Then, m lines follow corresponding to each clause. Each clause is a disjunction of literals in the form Xi or ~Xi for some $1 \leq i \leq n$, where ~Xi indicates the negation of the literal Xi. The "or" operator is denoted by a 'v' character and is seperated from literals with a single space.

Output

For each test case, display `satisfiable` on a single line if there is a satisfiable assignment; otherwise display `unsatisfiable`.

Sample Input 1

```
2
3 3
X1 v X2
~X1
~X2 v X3
3 5
X1 v X2 v X3
X1 v ~X2
X2 v ~X3
X3 v ~X1
~X1 v ~X2 v ~X3
```

Sample Output 1

```
satisfiable
unsatisfiable
```

Difficulty: 3.9

Download:
Sample data files

Author: Babak Behsaz

Source: Rocky Mountain Regional Contest (RMRC) 2014

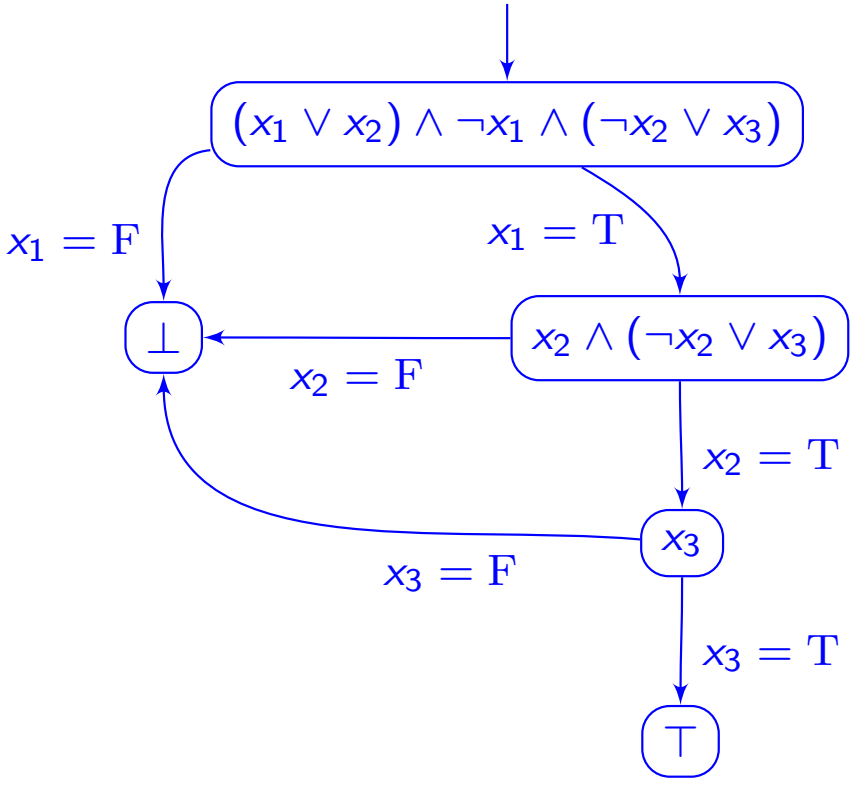
License: (cc) BY-SA

Satisfiability test-input 1(a)

Noder: logiska formler
Kanter: tilldelning till variabler

Vi bygger ingen explicit graf;
implicit sökning över formler
För n variables upp till 2^n noder

Val av "tillåtna" tilldelningar här godtyckligt. Kunde ha val mellan $x_2 = \mathrm{T}$, $x_3 = \mathrm{F}$, eller fler alternativ, vid roten.



noder	*kanter*
konfigurationer	*transitioner*
tillstånd	*transitioner*

Vi kommer återbesöka denna typ av problem (Davis-Putnam etc.)

Sök-repetition: skelettet

Liknande struktur på sökalgoritmerna, många på formen att givet en lämplig arbets-samling D och em resultat-datastruktur R:

```
put(D,starting_node)
initialisera R
while len(D)>0:
    v=take(D)
    for e in edges_from(v):
        if R påstår att e är relevant:
            uppdatera R
            put(D,target(e))
return R
```

Sök-repetition: depth-first

"Folklore" alt. Konrad Zuse 1945

D en stack, R vilka noder som redan hittats

```
stack=[]
stack.append(starting_node)
seen=set()
while len(stack)>0:
    v=stack.pop()
    for e in edges_from(v):
        if not target(e) in seen:
            seen[target(e)]=True
            stack.append(target(e))
return seen
```

Enklast och därför standard i många sammanhang

Prolog kombinerar djupet-först-sökning med unifikation till ett helt programmeringsspråk

Sök-repetition: depth-first (rekursiv)

Alternativt rekursion

```
# Returns true if v is reachable from u
def dfs(u,v,seen):
    if u==v:
        return True
    # If we've tried u before don't bother
    if u in seen:
        return False
    else:
        seen.add(u)
        for e in edges_from(u):
            if dfs(target(e),seen):
                return True
    return False

dfs(starting_node,set())
```

Potentiellt problem med overflow etc.

Sök-repetition: breadth-first

"Folklore" alt. Konrad Zuse 1945

D en kö, R vilka noder som redan hittats

```
queue=deque()
queue.append(starting_node)
seen=set()
while len(queue)>0:
    v=queue.popleft()
    for e in edges_from(v):
        if not target(e) in seen:
            seen[target(e)]=True
            queue.append(target(e))
return seen
```

Kortaste vägen utan vikter, "flood-fill". Kanren och core.logic kombinerar bredden-först med unifikation för "modernare" logik-lösare

Sök-repetition: breadth-first (med vettigt output)

Mer saker i *R* oftast, t.ex. vägen man når en nod, `prev`

```
queue=deque()
queue.append(starting_node)
seen=set()
prev={}
while len(queue)>0:
    v=queue.popleft()
    for e in edges_from(v):
        if not target(e) in seen:
            prev[target(e)]=source(e)
            seen[target(e)]=True
            queue.append(target(e))
return prev
```

Hur tar man sig till *v* från start-noden?

```
prev=dfs(starting_node)
path_to_v=[v]
while v!=u:
    path_to_v.append(prev[v])
    v=prev[v]
path_to_v.reverse()  # Or do by loop
```

Dijkstra's algoritm: single-source shortest path

Några extra detaljer i strukturen, men D är en heap och R avstånd och hur vi nått till noder:

```
dist[start]=0
heap=[(0,start)]
seen=set()

while len(heap)>0 and len(seen)<len(all_nodes):
    _,v=heapq.heappop(heap)
    if v in seen:
        continue
    seen.add(v)
    for e in edges_from(v):
        if dist[v]+weight<dist[target(e)]:
            prev[target(e)]=[v]
            dist[target(e)]=dist[v]+weight
            heapq.heappush(heap,(dist[v]+weight,v))
```

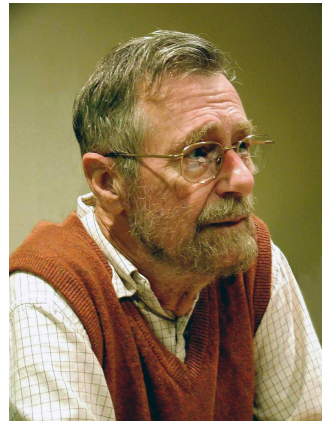
Edsger Dijkstra

1959

Dijkstra's algoritm: single-source shortest path

Några extra detaljer i strukturen, men D är en heap och R avstånd och hur vi nått till noder:

```
dist[start]=0
heap=[(0,start)]
seen=set()

while len(heap)>0 and len(seen)<len(all_nodes):
    _,v=heapq.heappop(heap)
    if v in seen:
        continue
    seen.add(v)
    for e in edges_from(v):
        if dist[v]+weight<dist[target(e)]:
            prev[target(e)]=[v]
            dist[target(e)]=dist[v]+weight
            heapq.heappush(heap,(dist[v]+weight,v))
```



Edsger Dijkstra

1959

Som här skriven blir den $\mathcal{O}(|E| \log |E|)$, andra alternativ:

- implementera $\mathcal{O}(\log n)$ *decrease-key* för $\mathcal{O}(|E| \log |V|)$,
- implementera en Fibonacci heap för $\mathcal{O}(|E| + |V| \log |V|)$.

Fountain

Consider a grid consisting of N rows and M columns, where each cell is either air, stone, or water. Each second, the water spreads in the following fashion:

1. If a water cell is directly above an air cell then the air cell turns into water in the next second.
2. If a water cell is directly above a stone cell then any air cells directly left or right of the water cell turn into water in the next second.

After some number of seconds, the water will have stopped spreading. Show how the grid looks when that happens. You can assume that all cells outside of the grid behave as air cells; for instance, if a water cell is at the bottommost row then its water will not spread to the sides.

Input

The first line consists of two integers N and M ($2 \leq N, M \leq 50$), the number of rows and columns in the grid.

Each of the following N lines contains a string S of length M. The string S represents one of the rows in the grid. It consists of the symbols "`.`" (air), "`#`" (stone), and "`V`" (water).

Output

Print N lines, each consisting of a string of length M, describing the grid as it looks when the water has stopped spreading.

Sample Input 1

```
5 7
...V...
.......
.......
...#...
..###..
```

Sample Output 1

```
...V...
...V...
..VVV..
.VV#VV.
.V###V.
```

Submit | Stats

Problem ID: fontan

CPU Time limit: 1 second

Memory limit: 1024 MB

Difficulty: 2.3

Language: en, sv

Download:
Sample data files

Author: Nils Gustafsson

Source: Kodsport Summer Challenge 2020

You can assume that all cells outside of the grid behave as air cells; for instance, if a water cell is at the bottommost row then its water will not spread to the sides.

Input

The first line consists of two integers N and M ($2 \leq N, M \leq 50$), the number of rows and columns in the grid.

Each of the following N lines contains a string S of length M. The string S represents one of the rows in the grid. It consists of the symbols "`.`" (air), "`#`" (stone), and "`V`" (water).

Output

Print N lines, each consisting of a string of length M, describing the grid as it looks when the water has stopped spreading.

Sample Input 1

```
5 7
...V...
.......
.......
...#...
..###..
```

Sample Output 1

```
...V...
...V...
..VVV..
.VV#VV.
.V###V.
```

Sample Input 2

```
12 14
....V...V.....
..............
..............
..............
...#########..
.......#......
.......#.#....
.......#####..
........###...
........#.#...
........#.#...
##############
```

Sample Output 2

```
....V...V.....
....V...V.....
....V...V.....
..VVVVVVVVVV.
..V#########V.
..V....#....V.
..V....#.#..V.
..V....#####V.
..V.....###.V.
..V.....#.#.V.
VVVVVVVV#.#VVV
##############
```

Author: Nils Gustafsson

Source: Kodsport Summer Challenge 2020

Låt oss lösa ett problem

Bredden-först-sökning.

Noder: Cellerna i diagrammet, koordinater (x, y).
Kanter: Varje nod har antingen kant till noden för *cellen nedanför* (om den *inte är en sten)* eller till noderna för *cellerna till vänster och höger* (om den nedanför *är en sten).*

Notera att det är onödigt besvärligt att konstruera en abstrakt graf (och således är knappast ett bibliotek aktuellt)

Lite preparation

```
# Format input
grid=[list(l.strip()) for l in inp[1:]]
width=len(grid[0])
height=len(grid)

# Get cell contents, returning 'o' if outside the grid
def get(x,y):
    if x<0 or x>=width or y<0 and y>=height:
        return 'o'
    else:
        return grid[y][x]

# BFS queue
queue=deque()

# Initialize with all cells currently containing water
for x in range(width):
    for y in range(height):
        if get(x,y)=='V':
            queue.append((x,y))
```

Sedan bredden först

```
while len(queue)>0:
    x,y=queue.popleft()
    below=get(x,y+1)
    if below=='#':  # rock
        if get(x-1,y)=='.':
            grid[y][x-1]='V'
            queue.append((x-1,y))
        if get(x+1,y)=='.':
            grid[y][x+1]='V'
            queue.append((x+1,y))
    elif below=='.': # air
        grid[y+1][x]='V'
        queue.append((x,y+1))

for y in range(height):
    for x in range(width):
        print(f'{grid[y][x]}',end='')
    print('')
```

"Resultatet" R nu aktuellt grid, `seen=set()` ersatt av att uppdatera celler: är ändå både input och output

Sedan bredden först

```
while len(queue)>0:
    x,y=queue.popleft()
    below=get(x,y+1)
    if below=='#':  # rock
        if get(x-1,y)=='.':
            grid[y][x-1]='V'
            queue.append((x-1,y))
        if get(x+1,y)=='.':
            grid[y][x+1]='V'
            queue.append((x+1,y))
    elif below=='.': # air
        grid[y+1][x]='V'
        queue.append((x,y+1))

for y in range(height):
    for x in range(width):
        print(f'{grid[y][x]}',end='')
    print('')
```

Undvika explicit graf är inte bara lättja i leksaksproblem: sök över 5 terabyte finansdata lagrat i en SQL-server

Hoppa Hage

Det finns en ny konstinstallation i stan, och den inspirerar dig till en lek! Konstinstallationen är ett golv med en $n \times n$ matris av fyrkantiga plattor. Varje platta innehåller ett nummer mellan 1 och k. Leken är nu att hoppa hage på den, du skall börja på en platta markerad 1, hoppa till en platta numrerad 2, sedan 3, och så vidare, tills du når en platta markerad k. Du kommer alltså i slutet av sekvensen ha besökt exakt en platta med varje nummer. Då du är en mycket bra hoppare kan du nå vilken platta som helst i ett enda skutt.
Räkna avståndet mellan plattorna med Manhattan-avstånd (så avståndet mellan (x_1, y_1) och (x_2, y_2) är $|x_2 - x_1| + |y_2 - y_1|$). Vad är det *kortaste totala avståndet* du behöver hoppa för att genomföra leken?

Input: Första raden består av två heltal, $1 \leq n \leq 100$ och $1 \leq k \leq n^2$, där konstinstallationen består i en $n \times n$ matris av heltal från 1 till k. Var och en av de nästa n raderna innehåller n heltal mellan 1 och k. Detta är konstinstallationen.

Output: Ett heltal, längden på den kortaste hag-hoppnings-sekvensen som startar på en platta markerad 1 och slutar på en platta markerad k. Skriv ut -1 om ingen tillåten sekvens existerar.

Exempel på input och output:

Input	Output
3 5 1 2 3 5 5 4 1 1 1	4

Input	Output
4 6 1 6 6 6 6 2 6 6 6 5 3 6 1 5 1 4	10

Input	Output
3 5 1 3 5 1 3 2 5 2 3	-1

Intercept

Fatima commutes from KTH to home by subway every day. Today Robert decided to surprise Fatima by baking cookies and bringing them to an intermediate station. Fatima does not always take the same route home, because she loves to admire the artwork inside different stations in Stockholm. However, she always optimizes her travel by taking the shortest route. Can you tell Robert which station he should go to in order to surely intercept Fatima?

Photo by Patrik Neckman

Input

The first line contains two integers N and M, $1 \leq N, M \leq 100\,000$, where N is the number of subway stations and M is the number of subway links. M lines follow, each with three integers u, v, w, $0 \leq u, v < N$, $0 < w \leq 1\,000\,000\,000$, meaning that there is a one-way link from u to v that takes w seconds to complete. Note that different subway lines may serve the same route.

The last line contains two integers s and t, $0 \leq s, t < N$ the number of the station closest to KTH and closest to home, respectively. It is possible to reach t from s.

Output

A space separated list of all the station numbers u such that all shortest paths from s to t pass through u, in increasing order.

Sample Input 1

```
4 4
0 1 100
0 2 100
1 3 100
2 3 100
0 3
```

Sample Output 1

```
0 3
```

Submit | Stats

My Submissions

Problem ID: intercept

CPU Time limit: 4 seconds

Memory limit: 1024 MB

Difficulty: 6.7

Download:

Sample data files

Author: Marc Vinyals

Source: KTH Challenge 2014

Photo by Patrik Neckman

Input

The first line contains two integers N and M, $1 \leq N, M \leq 100\,000$, where N is the number of subway stations and M is the number of subway links. M lines follow, each with three integers u, v, w, $0 \leq u, v < N$, $0 < w \leq 1\,000\,000\,000$, meaning that there is a one-way link from u to v that takes w seconds to complete. Note that different subway lines may serve the same route.

The last line contains two integers s and t, $0 \leq s, t < N$ the number of the station closest to KTH and closest to home, respectively. It is possible to reach t from s.

Output

A space separated list of all the station numbers u such that all shortest paths from s to t pass through u, in increasing order.

Sample Input 1

```
4 4
0 1 100
0 2 100
1 3 100
2 3 100
0 3
```

Sample Output 1

```
0 3
```

Sample Input 2

```
7 8
0 1 100
0 2 100
1 3 100
2 3 100
3 4 100
3 5 100
4 6 100
5 6 100
0 6
```

Sample Output 2

```
0 3 6
```

Author: Marc Vinyals

Source: KTH Challenge 2014

Lösningens delar

Svårt för kursen! Tunnelbanegraf G: *sök, sök, och sök lite mer*

1. Vilka är *alla* de kortaste vägarna?
   - Måste beskrivas kompakt, potentiellt exponentiellt många

Lösningens delar

Svårt för kursen! Tunnelbanegraf G: *sök, sök, och sök lite mer*

1. Vilka är *alla* de kortaste vägarna?
   - Måste beskrivas kompakt, potentiellt exponentiellt många
   - $w > 0$, inga cykler med 0 eller negativ vikt: ett *urval av kanter i G* bildar en *riktad acyklisk graf* där alla vägar från start till slut har samma minimala vikt.
   - Vi kan modifiera Dijkstra's till att hitta denna!

Lösningens delar

Svårt för kursen! Tunnelbanegraf G: *sök, sök, och sök lite mer*

1. Vilka är *alla* de kortaste vägarna?
   - Måste beskrivas kompakt, potentiellt exponentiellt många
   - $w > 0$, inga cykler med 0 eller negativ vikt: ett *urval av kanter i G* bildar en *riktad acyklisk graf* där alla vägar från start till slut har samma minimala vikt.
   - Vi kan modifiera Dijkstra's till att hitta denna!
2. På en riktad acyklisk graf: hitta noderna som alla vägar från start till mål passerar igenom
   - Fortfarande exponentiellt många vägar
   - I flödes-termer: hitta alla noder som *dominerar* målet, för alla noder v (med s startnoden):

$$\mathrm{dom}(v) = \begin{cases} \{s\} & \text{when } v = s, \\ \{v\} \cup \bigcap_{v' \in \mathsf{next}[v]} \mathrm{dom}(v') & \text{otherwise,} \end{cases}$$

   så är $\mathrm{dom}(t)$ svaret.

Lösningens delar

Svårt för kursen! Tunnelbanegraf G: *sök, sök, och sök lite mer*

1. Vilka är *alla* de kortaste vägarna?
   - Måste beskrivas kompakt, potentiellt exponentiellt många
   - $w > 0$, inga cykler med 0 eller negativ vikt: ett *urval av kanter i G* bildar en *riktad acyklisk graf* där alla vägar från start till slut har samma minimala vikt.
   - Vi kan modifiera Dijkstra's till att hitta denna!
2. På en riktad acyklisk graf: hitta noderna som alla vägar från start till mål passerar igenom
   - Fortfarande exponentiellt många vägar
   - I flödes-termer: hitta alla noder som *dominerar* starten, för alla noder v (med t målnoden):

$$\mathrm{dom}(v) = \begin{cases} \{s\} & \text{when } v = s, \\ \{v\} \cup \bigcap_{v' \in \mathsf{prev}[v]} \mathrm{dom}(v') & \text{otherwise,} \end{cases}$$

   så är $\mathrm{dom}(s)$ svaret. *Ekvivalent baklänges.*

Oläsbar input-parsning

```
nnodes,_=inp[0]
start,goal=inp[-1]
routes=[[]]*nnodes
for k,v in groupby(sorted(inp[1:-1]),lambda x: x[0]):
    routes[k]=[(d[1],d[2]) for d in v]
```

Så *routes[1]=[(2,100),(5,50)]* om det finns en 100 sekunders resa från station 1 till 2, och en 50 sekunders resa från 1 till 5.

Dijkstra's med acyklisk kortaste-vägar graf ut

```
dist=[2**62]*nnodes
prev=[[]]*nnodes
dist[start]=0
seen=set()
heap=[(0,start)]

# Construct the closest-paths almost-DAG
while not goal in seen:
    _,source=heapq.heappop(heap)
    if source in seen:
        continue
    seen.add(source)
    for target,weight in routes[source]:
        new_dist=dist[source]+weight
        if new_dist==dist[target]:
            prev[target].append(source)
        elif new_dist<dist[target]:
            prev[target]=[source]
            dist[target]=new_dist
            heapq.heappush(heap,(new_dist,target))
```

Första akten

Min *ursprungliga* lösning: krånglig!

(Snabb titt, vi återvänder om vi har tid)

Beräkna dominerande noder: topologisk ordning

Nu beskriver `prev[]` en riktad acyklisk graf (DAG) med minimala vägar från t till s

För att beräkna $\mathrm{dom}(v)$ måste vi ha $\mathrm{dom}(v')$ för alla v' med en kant till v, vi sorterar vår DAG *topologiskt:*

Definition

För en acyklisk graf $G = (V, E)$ är $V = \{v_1, \ldots, v_n\}$ en *topologisk ordning* om alla kanter $E = (v_i, v_j)$ har $i < j$.

Beräkna dominerande noder: topologisk ordning

Nu beskriver `prev[]` en riktad acyklisk graf (DAG) med minimala vägar från t till s

För att beräkna $\mathrm{dom}(v)$ måste vi ha $\mathrm{dom}(v')$ för alla v' med en kant till v, vi sorterar vår DAG *topologiskt:*

Definition

För en acyklisk graf $G = (V, E)$ är $V = \{v_1, \ldots, v_n\}$ en *topologisk ordning* om alla kanter $E = (v_i, v_j)$ har $i < j$.

```
def topo(current_node,seen,nodes_in_topo):
    seen.add(current_node)
    for next_node in prev[current_node]:
        if not next_node in seen:
            topo(next_node,seen,nodes_in_topo)
    nodes_in_topo.append(current_node)
    return nodes_in_topo
topo(goal,nodes_in_topological_order) # Sort topological
```

Beräkna dominerande noder: topologisk ordning

Nu beskriver `prev[]` en riktad acyklisk graf (DAG) med minimala vägar från t till s

För att beräkna $\mathrm{dom}(v)$ måste vi ha $\mathrm{dom}(v')$ för alla v' med en kant till v, vi sorterar vår DAG *topologiskt:*

Definition

För en acyklisk graf $G = (V, E)$ är $V = \{v_1, \ldots, v_n\}$ en *topologisk ordning* om alla kanter $E = (v_i, v_j)$ har $i < j$.

```
def topo(current_node,seen,nodes_in_topo):
    seen.add(current_node)
    for next_node in prev[current_node]:
```

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8972100	2022-05-20 17:05:26	Intercept	✖ Run Time Error	0.27 s	Python 3

```
topo(goal,nodes_in_topological_order) # Sort topological
```

Beräkna dominerande noder: topologisk ordning

Nu beskriver `prev[]` en riktad acyklisk graf (DAG) med minimala vägar från t till s

För att beräkna $\mathrm{dom}(v)$ måste vi ha $\mathrm{dom}(v')$ för alla v' med en kant till v, vi sorterar vår DAG *topologiskt:*

Definition

För en acyklisk graf $G = (V, E)$ är $V = \{v_1, \ldots, v_n\}$ en *topologisk ordning* om alla kanter $E = (v_i, v_j)$ har $i < j$.

```
def topo(current_node,seen,nodes_in_topo):
    seen.add(current_node)
    for next_node in prev[current_node]:
```

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8972100	2022-05-20 17:05:26	Intercept	✖ Run Time Error	0.27 s	Python 3

```
topo(goal,nodes_in_topological_order) # Sort topological

Traceback: (topo*16,000)..
RecursionError: maximum recursion depth exceeded
```

Beräkna dominerande noder: topologisk ordning

```
OP_ADD_TS=0
OP_DO_DFS=1
stack=[]
stack.append((OP_ADD_TS,goal))
stack.append((OP_DO_DFS,goal))
seen=set()
ts=[]
while len(stack)>0:
    op,v=stack.pop()
    if op==OP_ADD_TS:
        ts.append(v)
        continue
    for v_to in prev[v]:
        if not v_to in seen:
            seen.add(v_to)
            stack.append((OP_ADD_TS,v_to))
            stack.append((OP_DO_DFS,v_to))
ts=ts[::-1] # We generated this backwards
```

Dominerande noder när man tänker för kort

```
dom=[set(range(nnodes)) for i in range(nnodes)]
dom[goal]=set()
for v in ts:
    dom[v].add(v) # Nodes always dominate themselves
    for v_to in prev[v]:
        dom[v_to]=dom[v_to]&dom[v]
print("␣".join([str(x) for x in sorted(dom[start])]))
```

Helt korrekt! Men...

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8971682	2022-05-23 14:51:52	Intercept	✖ **Memory Limit Exceeded**	1.83 s	Python 3

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8982062	2022-05-23 15:00:52	Intercept	✖ **Time Limit Exceeded**	> 4.00 s	Python 3

Dominerande noder när man tänker för kort

```
dom=[set(range(nnodes)) for i in range(nnodes)]
dom[goal]=set()
for v in ts:
    dom[v].add(v) # Nodes always dominate themselves
    for v_to in prev[v]:
        dom[v_to]=dom[v_to]&dom[v]
print("␣".join([str(x) for x in sorted(dom[start])]))
```

Helt korrekt! Men $\mathcal{O}(n^2)$ både tid och minne!
$(10^5)^2 = 10{,}000{,}000{,}000$, helt klart inte med 1 gigabyte minne, troligen inte heller på 4 sekunder (åtminstone inte i Python)

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8971682	2022-05-23 14:51:52	Intercept	✖ Memory Limit Exceeded	1.83 s	Python 3

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8982062	2022-05-23 15:00:52	Intercept	✖ Time Limit Exceeded	> 4.00 s	Python 3

Dominerande noder: låt oss tänka om

Fix: bygg en "lat" representation och evaluera i ett svep? T.ex.

```
(UNION,(VAL,12),(INTERSECT,(UNION,(VAL,50),(VAL,11)),...
```

Bör fungera, och jag slösade massor med tid på det[1].

[1] I mitt försvar: jag utesluter lösningar med kod för lång för slides

Dominerande noder: låt oss tänka om

Fix: bygg en "lat" representation och evaluera i ett svep? T.ex.

```
(UNION,(VAL,12),(INTERSECT,(UNION,(VAL,50),(VAL,11)),...
```

Bör fungera, och jag slösade massor med tid på det[1].

Full domineringsrelation är onödigt generell, *ny ansats:* flödesberäkning ("network flow"), jämför elektricitet

1. Skicka en enhet "flöde" till slutnoden, `flow[t]=1`
2. När en nod *v* fått allt sitt inkommande flöde (topologisk ordning!), skicka `flow[v]/len(prev[v])` till nod i `prev`, dvs "föräldrarna"
3. Alla noder *v* med en enhet flöde, `flow[v]==1`, dominerar

Var försiktig med precision: tillräckligt "breda" DAGs kommer att förlora flöde med vanliga flyttal

I Python: `from fractions import Fraction`

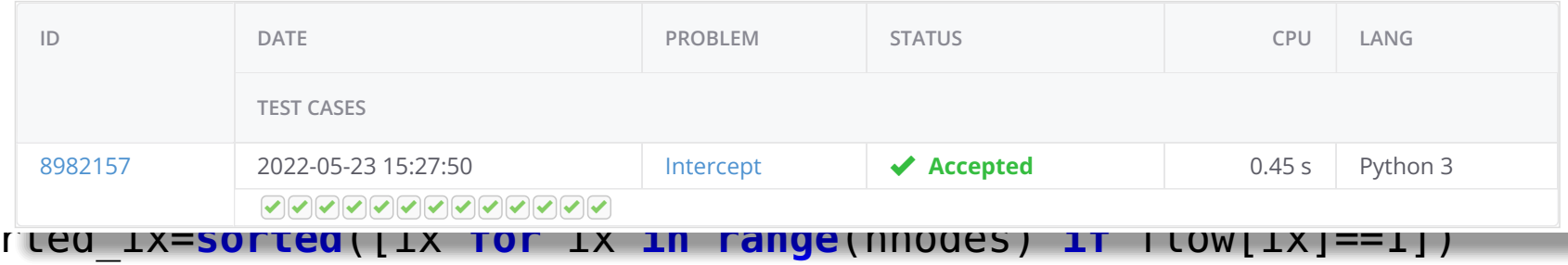
[1] I mitt försvar: jag utesluter lösningar med kod för lång för slides

Flödesberäkning för dominerande noder

```
flow=[Fraction(0)]*nnodes
flow[goal]=Fraction(1)
for v in ts:
    if len(prev[v])==0:
        break # Divide by zero dodge (always the last node)
    cflow=flow[v]/len(prev[v])
    for v_to in prev[v]:
        flow[v_to]+=cflow

sorted_ix=sorted([ix for ix in range(nnodes) if flow[ix]==1])
print("␣".join([str(x) for x in sorted_ix]))
```

Flödesberäkning för dominerande noder

```
flow=[Fraction(0)]*nnodes
flow[goal]=Fraction(1)
for v in ts:
    if len(prev[v])==0:
        break # Divide by zero dodge (always the last node)
```



ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8982157	2022-05-23 15:27:50	Intercept	✔ Accepted	0.45 s	Python 3

```
sorted_ix=sorted([ix for ix in range(nnodes) if flow[ix]==1])
print("␣".join([str(x) for x in sorted_ix]))
```

För att sammanfatta: Dijkstra's modifierad att producera en DAG, djupet-först modifierad att producera en topologisk ordning, sedan topologisk promenad för flöde.

Andra akten

Skiena's tankar om *articulating nodes*

(Också väl krångligt)

Djupet först på Skiena's sätt

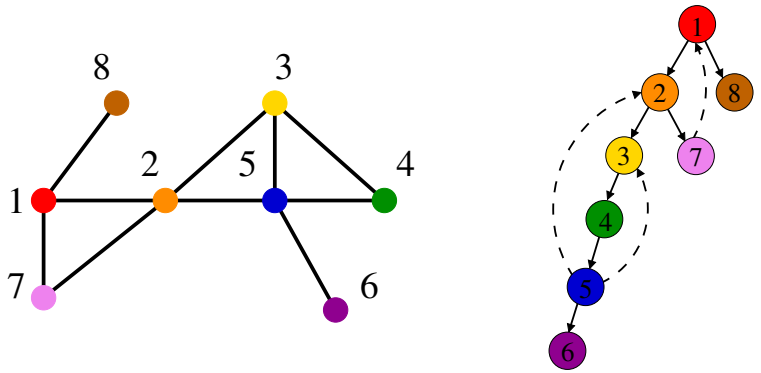
DFS has a neat recursive implementation which eliminates the need to explicitly use a stack.
Discovery and final times are a convenience to maintain.

```
void dfs(graph *g, int v) {
    edgenode *p;          /* temporary pointer */
    int y;                /* successor vertex */

    if (finished) {
        return;           /* allow for search termination */
    }

    discovered[v] = true;
    time = time + 1;
    entry_time[v] = time;

    process_vertex_early(v);

    p = g->edges[v];
    while (p != NULL) {
        y = p->y;
        if (!discovered[y]) {
```

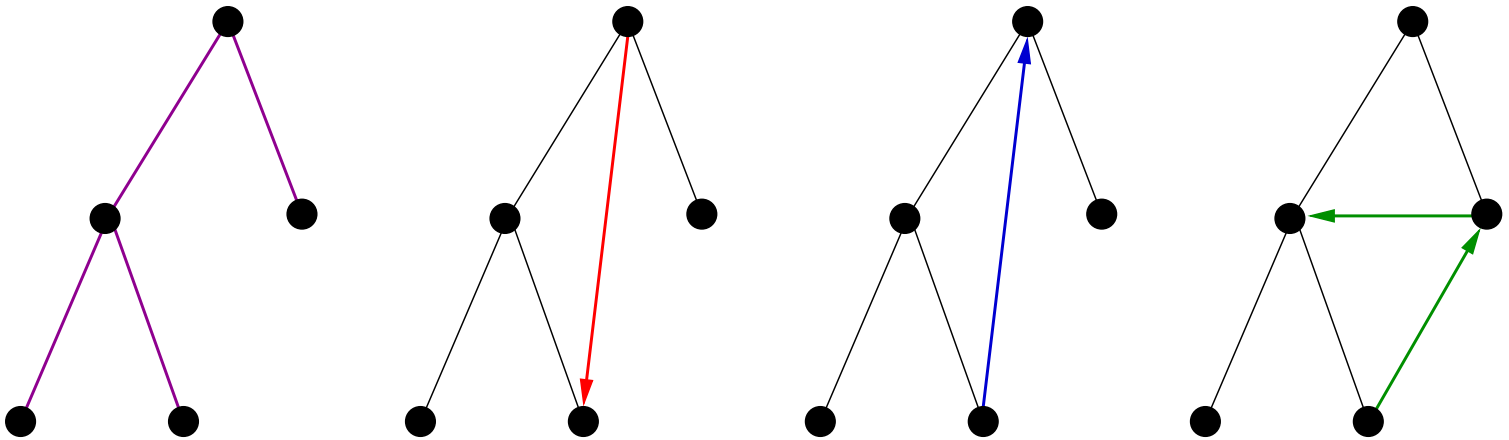
Djupet först på Skiena's sätt

```
        } else if (((!processed[y]) && (parent[v] != y)) || (g->directed)) {
            process_edge(v, y);
        }

        if (finished) {
            return;
        }

        p = p->next;
    }

    process_vertex_late(v);

    time = time + 1;
    exit_time[v] = time;

    processed[v] = true;
}
```

A depth-first search of a graph organizes the edges of the graph in a precise way.
In a DFS of an undirected graph, we assign a direction to each edge, from the vertex which discover it:



Every edge is either:



On any particular DFS or BFS of a directed or undirected graph, each edge gets classified as one of the above.

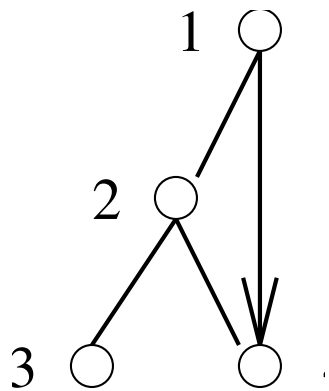
Kant-klasser

```
int edge_classification(int x, int y) {
    if (parent[y] == x) {
        return(TREE);
    }

    if (discovered[y] && !processed[y]) {
        return(BACK);
    }

    if (processed[y] && (entry_time[y]>entry_time[x])) {
        return(FORWARD);
    }

    if (processed[y] && (entry_time[y]<entry_time[x])) {
        return(CROSS);
    }

    printf("Warning: self loop (%d,%d)\n", x, y);

    return -1;
}
```

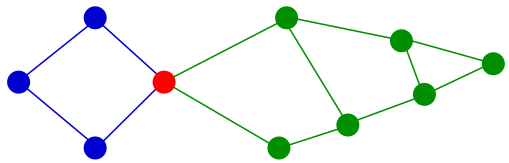
The reason DFS is so important is that it defines a very nice ordering to the edges of the graph.

In a DFS of an undirected graph, every edge is either a tree edge or a back edge.

Why? Suppose we have a forward edge. We would have encountered $(4, 1)$ when expanding 4, so this is a back edge.



Suppose you are a terrorist, seeking to disrupt the telephone network. Which station do you blow up?



An *articulation vertex* is a vertex of a connected graph whose deletion disconnects the graph.
Clearly connectivity is an important concern in the design of any network.
Articulation vertices can be found in $O(n(m+n))$ – just delete each vertex to do a DFS on the remaining graph to see if it is connected.

Kant-klasser i DFS

In a DFS tree, a vertex v (other than the root) is an articulation vertex iff v is not a leaf and some subtree of v has no back edge incident until a proper ancestor of v.