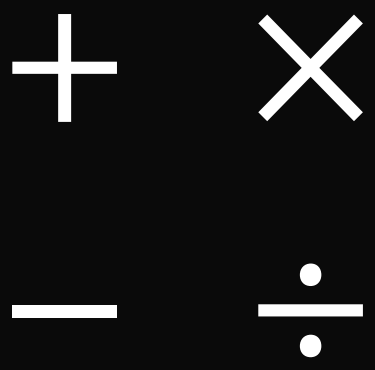


How to NP-Hardness

Informal notes, Thore Husfeldt, 27 October 2025

Here is a general template for communicating NP-hardness proofs. The aim is to be sufficiently strict so as to provide a scaffolding to express NP-hardness proofs in a way that maximises the chance of not overlooking anything and, in particular, make the reduction go 'in the right direction.'

There are many other ways of doing this.

Template

The template is here given for a reduction that shows that *Some Problem* (presumably a new problem, maybe one in an exercise or exam) is NP-hard by a reduction from *Other Problem* (presumably a well-known problem whose NP-hardness is already established.) In other words, we want to prove *Other Problem* $\leq_p$ *Some Problem*.

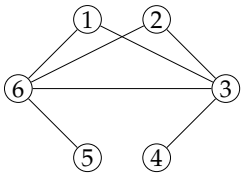
> **Claim:** *Some Problem* is NP-hard.
>
> *Proof.* We will reduce from *Other Problem*, which is known to be NP-hard.
>
> Let ... be an instance of *Other Problem.* (...)
>
> Construct and instance ... of *Some Problem* as follows. (...).
>
> For example, if ... is an instance to *Other Problem* then the resuling instance to *Some Problem* is ....
>
> (Note that the size of ... is polynomial in the size of ....)
>
> Now let ... be a solution to *Some Problem* for the instance ....
> Conversely, if ... is a solution to *Other Problem*, then ....

Example

Here is an example of the reduction Vertex Cover $\leq_p$ Set Cover follwing the template.

> **Claim:** Set Cover is NP-hard.
>
> *Proof.* We will reduce from Vertex Cover, which is known to be NP-hard.
>
> Let G, k be an instance of Vertex Cover. Enumerate the vertex set of G as $V = \{v_1, \ldots, v_n\}$. Enumerate the edge set of G as $E = \{e_1, \ldots, e_m\}$.
>
> Construct an instance U, F, k' of Set Cover as follows. Let $U = \{1, \ldots, m\}$ and construct the family $F = \{S_1, \ldots, S_n\}$ by setting S_i to be the indices of the edges incident on vertex v_i. Finally, set $k' = k$.

For instance, if G is the graph



then the resulting Set Cover instance is $U = \{1, \ldots, 7\}$ with

$$\begin{aligned} S_1 &= \{3, 7\}, \\ S_2 &= \{2, 4\}, \\ S_3 &= \{3, 4, 5, 6\}, \\ S_4 &= \{5\}, \\ S_5 &= \{1\}, \\ S_6 &= \{1, 2, 6, 7\}. \end{aligned}$$

(We note that the resulting instance has size $O(m + n)$, which is polynomial (in fact, linear) in the size of G.)

Now assume that $S \subseteq F$ is a solution to the Set Cover instance. In other words, the union of the k' sets in S contains all of U. Then the corresponding vertex subset in G, i.e., the set of vertices v_i for $S_i \in S$ is a vertex cover of G. Conversely, if $C \subseteq V$ is a vertex cover of size k then the corresponing subsets S_i for $v_i \in C$ constitute a set cover of U.

Russian Hacking

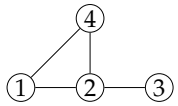
Here's an answer to an old exam (Russian Hacking) that would get full marks with a gold star:

Claim Russing Hacking is NP-hard.

Proof. We will reduce from Vertex Cover, which is known to be NP-hard.

Let G, k be an instance of vertex cover, with n vertices V and m edges E. Let's assume the vertices are named $\{1, \ldots, n\}$.

Constuct and instance of Russian Hacking as follows. There will be $R = n$ routers $1, \ldots, n$ and $W = m$ wires, one for every edge in E. Importantly, *each* of these wires is monitored. For instance, if G is



then the resulting instance to Russian Hacking is

```
4 4
1 -- 2 ?
2 -- 3 ?
2 -- 4 ?
1 -- 4 ?
```

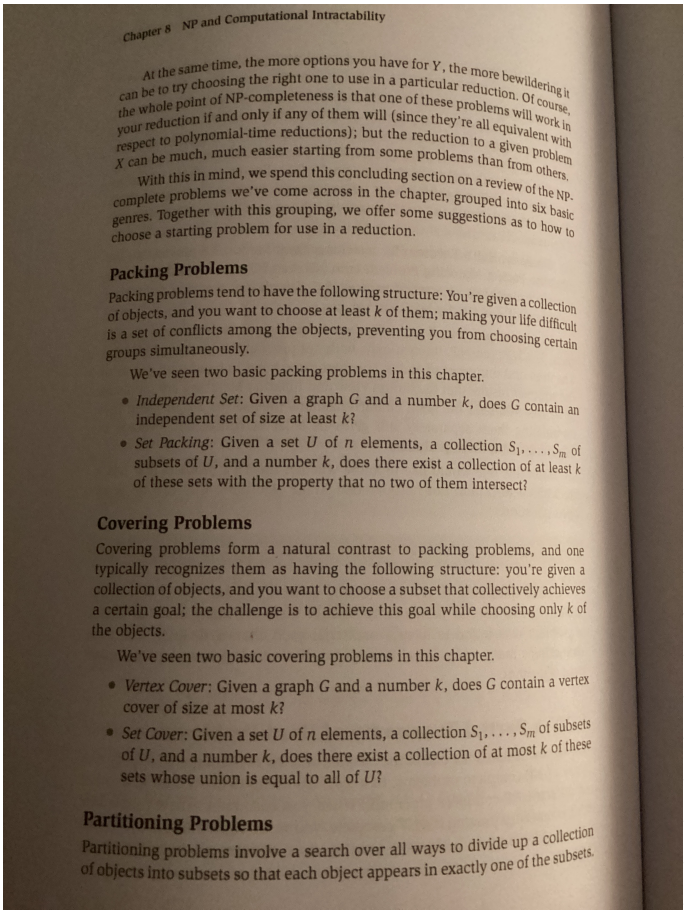
> Now assume there is a solution of size K to Russian Hacking consisting of the routers in R. Then R constitutes a minimal vertex cover in G (it covers every edge since every wire was monitored) of size K. Conversely, every vertex cover C of G of size k corresponds to a set of k many routers that cover every wire (because C covers every edge in G.)

Of course, the *core* of the argument could be communicated more briefly, as in

> "Reduce from vertex cover, vertices become routers and edges become wires, just remember to monitor *every* wire".

If you really know what you're doing and like to live dangerously, this single line probably gets you $\frac{2}{3}$ of the marks. Note the importance of the word "from" in the quoted sentence; if you write "to" you are likely to get 0 points because the argument is fundamentally flawed. (If you follow the template you won't make that mistake.)

NP Hard Problems



Chapter 8 NP and Computational Intractability

At the same time, the more options you have for Y, the more bewildering it can be to try choosing the right one to use in a particular reduction. Of course, the whole point of NP-completeness is that one of these problems will work in your reduction if and only if any of them will (since they're all equivalent with respect to polynomial-time reductions); but the reduction to a given problem X can be much, much easier starting from some problems than from others.

With this in mind, we spend this concluding section on a review of the NP-complete problems we've come across in the chapter, grouped into six basic genres. Together with this grouping, we offer some suggestions as to how to choose a starting problem for use in a reduction.

Packing Problems

Packing problems tend to have the following structure: You're given a collection of objects, and you want to choose at least k of them; making your life difficult is a set of conflicts among the objects, preventing you from choosing certain groups simultaneously.

We've seen two basic packing problems in this chapter.

- *Independent Set*: Given a graph G and a number k, does G contain an independent set of size at least k?
- *Set Packing*: Given a set U of n elements, a collection $S_1, \ldots, S_m$ of subsets of U, and a number k, does there exist a collection of at least k of these sets with the property that no two of them intersect?

Covering Problems

Covering problems form a natural contrast to packing problems, and one typically recognizes them as having the following structure: you're given a collection of objects, and you want to choose a subset that collectively achieves a certain goal; the challenge is to achieve this goal while choosing only k of the objects.

We've seen two basic covering problems in this chapter.

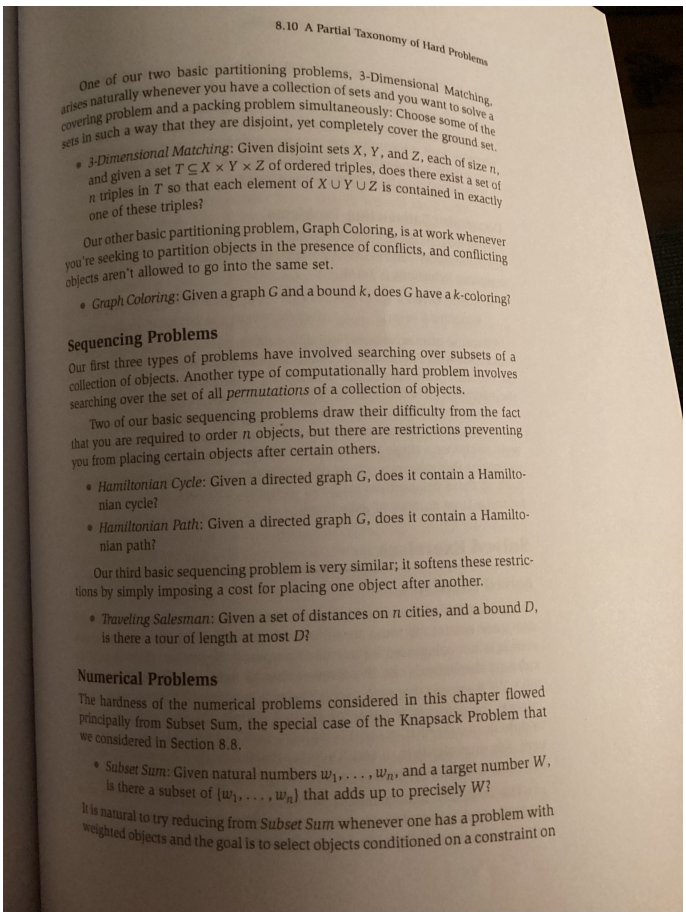
- *Vertex Cover*: Given a graph G and a number k, does G contain a vertex cover of size at most k?
- *Set Cover*: Given a set U of n elements, a collection $S_1, \ldots, S_m$ of subsets of U, and a number k, does there exist a collection of at most k of these sets whose union is equal to all of U?

Partitioning Problems

Partitioning problems involve a search over all ways to divide up a collection of objects into subsets so that each object appears in exactly one of the subsets.

Summation Formulas

$\sum_{k=1}^{n}(c) = nc$	$c + c + \cdots + c = nc$
$\sum_{k=1}^{n}(k) = \frac{n(n+1)}{2}$	$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$
$\sum_{k=1}^{n}(k^2) = \frac{n(n+1)(2n+1)}{6}$	$1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$
$\sum_{k=1}^{n}(k^3) = \left(\frac{n(n+1)}{2}\right)^2$	$1^3 + 2^3 + \cdots + n^3 = \left(\frac{n(n+1)}{2}\right)^2$
$\sum_{k=0}^{\infty}(x^k) = \frac{1}{1-x}$ for $\{-1 < x < 1\}$	$1 + x + x^2 + x^3 + x^4 + \cdots = \frac{1}{1-x}$
$\sum_{k=0}^{\infty}\left(\frac{x^k}{k!}\right) = e^x$	$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots = e^x$
$\sum_{k=1}^{\infty}\left[\frac{1}{k}\left(\frac{x-1}{x}\right)^k\right] = \ln x$ for $x \geq \frac{1}{2}$	$\left(\frac{x-1}{x}\right) + \frac{1}{2}\left(\frac{x-1}{x}\right)^2 + \frac{1}{3}\left(\frac{x-1}{x}\right)^3 + \cdots = \ln x$
$\sum_{k=1}^{\infty}\left[(-1)^{(k-1)}\left(\frac{x^k}{k}\right)\right] = \ln(1+x)$ for $\{-1 < x \leq 1\}$	$x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots = \ln(1+x)$
$\sum_{k=0}^{\infty}\left[(-1)^{(k)}\left(\frac{x^{(2k)}}{(2k)!}\right)\right] = \cos x$	$1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots = \cos x$
$\sum_{k=1}^{\infty}\left[(-1)^{(k-1)}\left(\frac{x^{(2k-1)}}{(2k-1)!}\right)\right] = \sin x$	$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \ldots = \sin x$



8.10 A Partial Taxonomy of Hard Problems

One of our two basic partitioning problems, 3-Dimensional Matching, arises naturally whenever you have a collection of sets and you want to solve a covering problem and a packing problem simultaneously: Choose some of the sets in such a way that they are disjoint, yet completely cover the ground set.

- *3-Dimensional Matching*: Given disjoint sets X, Y, and Z, each of size n, and given a set $T \subseteq X \times Y \times Z$ of ordered triples, does there exist a set of n triples in T so that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples?

Our other basic partitioning problem, Graph Coloring, is at work whenever you're seeking to partition objects in the presence of conflicts, and conflicting objects aren't allowed to go into the same set.

- *Graph Coloring*: Given a graph G and a bound k, does G have a k-coloring?

Sequencing Problems

Our first three types of problems have involved searching over subsets of a collection of objects. Another type of computationally hard problem involves searching over the set of all *permutations* of a collection of objects.

Two of our basic sequencing problems draw their difficulty from the fact that you are required to order n objects, but there are restrictions preventing you from placing certain objects after certain others.

- *Hamiltonian Cycle*: Given a directed graph G, does it contain a Hamiltonian cycle?
- *Hamiltonian Path*: Given a directed graph G, does it contain a Hamiltonian path?

Our third basic sequencing problem is very similar; it softens these restrictions by simply imposing a cost for placing one object after another.

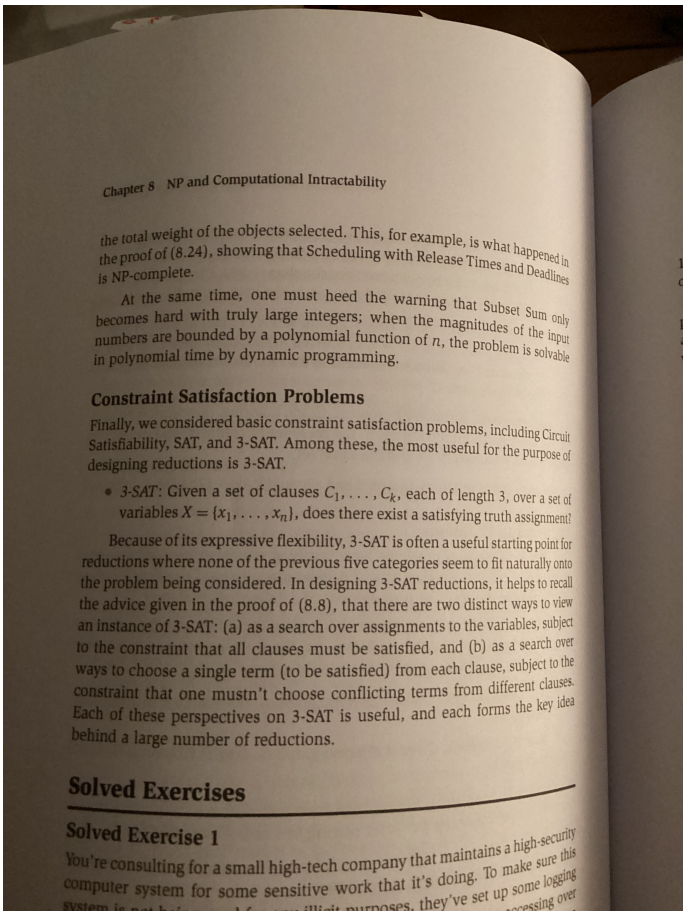
- *Traveling Salesman*: Given a set of distances on n cities, and a bound D, is there a tour of length at most D?

Numerical Problems

The hardness of the numerical problems considered in this chapter flowed principally from Subset Sum, the special case of the Knapsack Problem that we considered in Section 8.8.

- *Subset Sum*: Given natural numbers $w_1, \ldots, w_n$, and a target number W, is there a subset of $\{w_1, \ldots, w_n\}$ that adds up to precisely W?

It is natural to try reducing from *Subset Sum* whenever one has a problem with weighted objects and the goal is to select objects conditioned on a constraint on



Chapter 8 NP and Computational Intractability

the total weight of the objects selected. This, for example, is what happened in the proof of (8.24), showing that Scheduling with Release Times and Deadlines is NP-complete.

At the same time, one must heed the warning that Subset Sum only becomes hard with truly large integers; when the magnitudes of the input numbers are bounded by a polynomial function of n, the problem is solvable in polynomial time by dynamic programming.

Constraint Satisfaction Problems

Finally, we considered basic constraint satisfaction problems, including Circuit Satisfiability, SAT, and 3-SAT. Among these, the most useful for the purpose of designing reductions is 3-SAT.

- *3-SAT*: Given a set of clauses $C_1, \ldots, C_k$, each of length 3, over a set of variables $X = \{x_1, \ldots, x_n\}$, does there exist a satisfying truth assignment?

Because of its expressive flexibility, 3-SAT is often a useful starting point for reductions where none of the previous five categories seem to fit naturally onto the problem being considered. In designing 3-SAT reductions, it helps to recall the advice given in the proof of (8.8), that there are two distinct ways to view an instance of 3-SAT: (a) as a search over assignments to the variables, subject to the constraint that all clauses must be satisfied, and (b) as a search over ways to choose a single term (to be satisfied) from each clause, subject to the constraint that one mustn't choose conflicting terms from different clauses. Each of these perspectives on 3-SAT is useful, and each forms the key idea behind a large number of reductions.

Solved Exercises

Solved Exercise 1

You're consulting for a small high-tech company that maintains a high-security computer system for some sensitive work that it's doing. To make sure this system is not [illegible] purposes, they've set up some logging [illegible] processing over

How To Greedy

(a) Which one? → Just the name of the problem

(b) Describe algorithm Be careful & precise

Given Input read into format: ...

Sort the Input by X in asc/desc order.

Then do the following:

```
<result_tracker> = <default_value>
for n in Input:
    if <boolean_eval_of_n>:
        // Update result
        // Update state
    else:
        break // describe why all remaining n are not useful!
return / print <result_tracker>
```

this part may be different depending on the problem

(c) Running Time Make sure to describe all steps and substeps

Check n is length of input

Sorting takes $O(n \log n)$

Processing is $O(n)$, as the update of result and state may be done in constant time.

The Running time is therefore bounded by sorting at $O(n \log n)$

How To Divide-n-conquer

ⓐ which one? → just the name of the problem

ⓑ Describe algorithm

ⓒ Recurrence Relation

ⓓ Running Time

How To Graph Traversal

ⓐ which one? → just the name of the problem

ⓑ Describe the graph

The graph consists of x nodes, each representing X.

Edges in the graph represent Y and are (directed / undirected) (weighted / unweighted), with a total of y edges.

<Any other specification relevant to problem>

ⓒ Describe the algorithm

I use <algorithm name> on X to find Y. The result Y/? then represents S.

ⓓ Running Time

Breadth-First-Search (BFS) (p. 79 of textbook) $O(|V| + |E|)$
(1) Shortest path / reachability from s
(2) Connected component of s

Depth-First-Search (DFS) (p. 83 of textbook) $O(|V| + |E|)$
(1) Connected component of s

Topological Ordering (p. 102 of textbook) $O(|V| + |E|)$
(1) Define DAG order

Dijkstra's Algorithm (p. 180 of textbook) naive: $O(|V| \cdot |E|)$ HeapPQ: $O(|E| \cdot \log |V|)$
(1) Shortest paths in weighted graph

Prim's Algorithm (p. 191-192 of textbook) HeapPQ: $O(|E| \cdot \log |V|)$
(1) Minimum Spanning Tree (minimal total edge weight retaing connectivity)

Kruskal's Algorithm (p. 199 of textbook) $O(|E| \cdot \log |V|)$
(1) Minimum Spanning Tree
(2) Clustering (p. 201)

How To Dynamic Programming

@which one? → Just the name of the problem

(b) define OPT

if $w < w_i$ then $OPT(i,w) = OPT(i-1,w)$ ← Base case; we cannot afford w_i

Otherwise $OPT(i,w) = \max(OPT(i-1,w), w_i + OPT(i-1,w-w_i))$

did not take i; took i; optimization direction; agg depends on return

more base cases may exist: be complete

more cases may exist for subproblems

(c) Running Time and Space

Space: O(parameter space of OPT)

i.e. 1 parameter may be O(n), 2 may be O(n·m)...

Running Time: Same as Space, as we are filling out the lookup table

Backtracking to find S in solution matrix is O(n)

n: solution size

How To Network Flow

@ Which one? → Just the name of the problem

ⓑ Describe Reduction Be careful & precise

Create the source and sink nodes
Create a node for each X. Connect each node from X to Y by a (directed? undirected?) edge of capacity c.

...

The example Input may be transformed into the following graph G:

<DRAWING>

The max flow in this graph is equal to S which may be mapped to the solution space of problem by transformation(s).

The size of the graph G is $|V| = X + Y + 2$ (or more) $|E| = SRC_{edges} + SINK_{edges} + MIDDLE$ giving a total size of $O(|X| + |V|)$ (evaluate if this makes sense)

ⓒ Running Time

Ford-Fulkerson Algorithm (p. 344 of textbook) $O(|E| \cdot C)$
(1) Integer capacities only

Scaling Max-Flow (p. 353-354) $O(|E|^2 \log C)$
(1) Integer capacities only
(2) Good for graphs with high variance in capacities

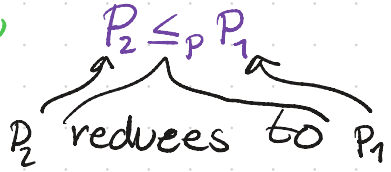
($m+n$) but $m \geq n/2$ as no singular nodes
max possible flow/capacity

How To NP-Hard

ⓐ which one? → Just the name of the problem

ⓑ which NP-Hard problem? → Find most similar problem, apply name

ⓒ Direction → Always known $\leq_p$ new

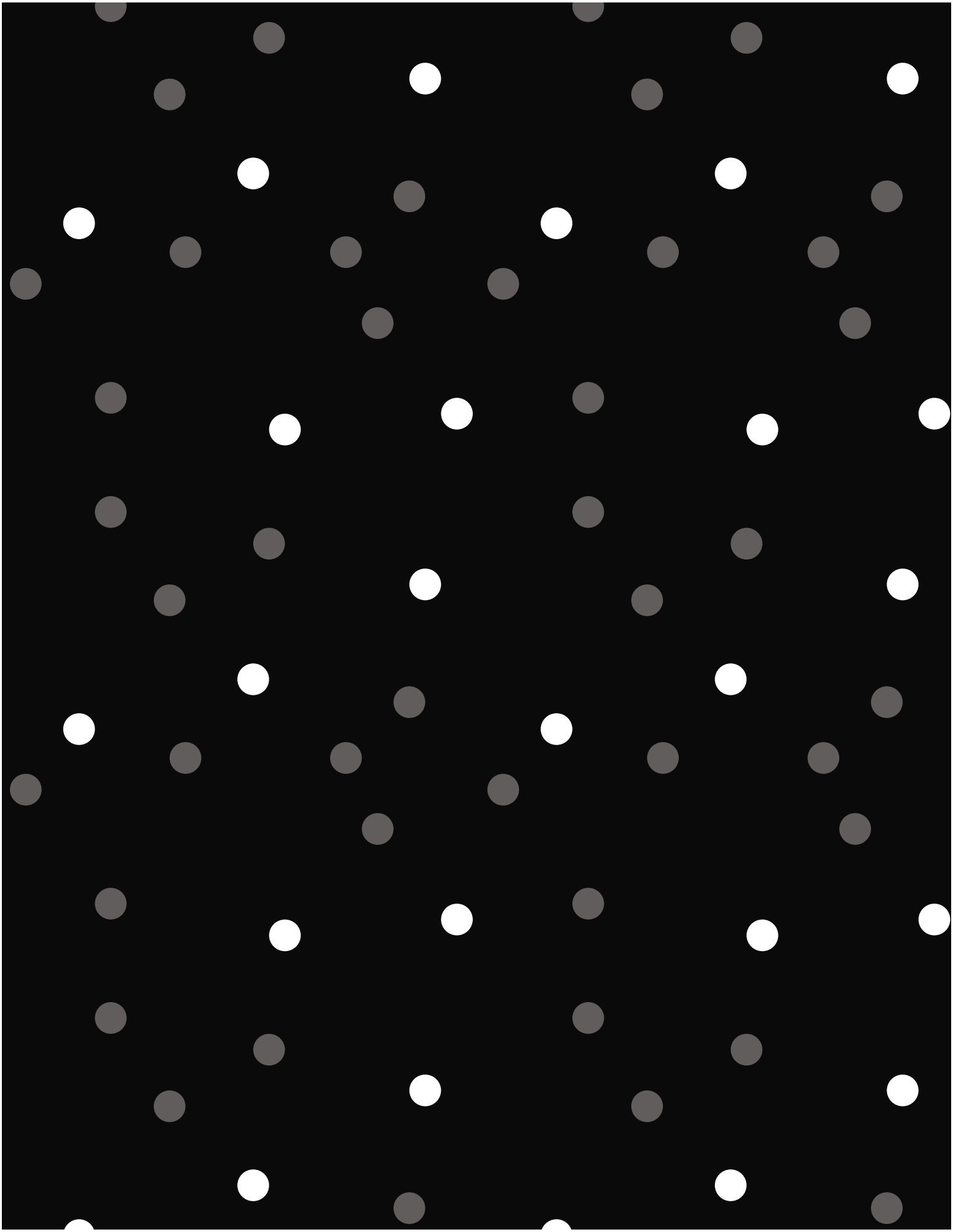


ⓓ Describe Reduction → Follow Template on page 2-4

① Pick Simple examples

② Describe "limitations" to problem mapping

③ Be careful & precise



Exam SAD1 4. April 2013

1 a) which is greedy

Zompf

b) describe algorithm

I will describe the algorithm for finding the solution in pseudocode as follows:

```
// each input entry is {"value": z_i, "index": i}
Coins ← input sorted by value in descending order

for i in [0, ..., k]:

    print coins[i]["index"] //print index of next most valuable
```

This pseudo-code will pick the k most valuable coins, assuming $n \geq k$.

c) Running time

Sorting the input is $O(n \log n)$
for loop with printing is $O(k)$

assuming $n \geq k$, we may say the loop is $O(n)$

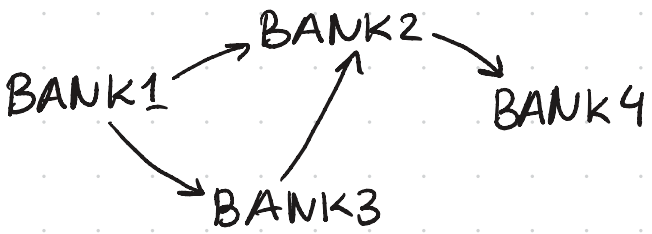
Thus the total runtime is $O(n \log n + n) = O(n \log n)$

2 a) which is graph connectivity?

Crisis

b) Describe graph

The graph is a directed graphs with each vertex being a bank and each edge indicating a dependency from the target on the source of the edge. $n \leq 2m$
There are m edges and up to $2m$ nodes edges are unweighted. Example instance:



c) which algorithm

Use BFS from the source node to determine if all nodes in the network are reachable

d) Running time

BFS is $O(\underbrace{2m}_{\text{nodes}} + \underbrace{m}_{\text{edges}}) = O(m)$

3

a) which is dynamic programming?

Neighbours

b) Give the reccurence relation for OPT

given a list of coins, enumerate them as $i \in [1, \dots, n]$

now let

$$OPT(i) = \begin{cases} [] & \text{if } i < 1 \\ [i] & \text{if } i == 1 \\ \text{longest}(OPT(i-1), OPT(i-2) + [i]) & \text{otherwise} \end{cases}$$

(+ : list append; $i-2$: skip neighbour)

c) Running time & space

The lookup table takes up space $O(n)$

Filling up the lookup table takes time $O(n)$

Backtracking is linear runtime $O(n)$, thus the total runtime is $O(n)$

4 a) which is network flow?

Lockout

b) Describe the reduction

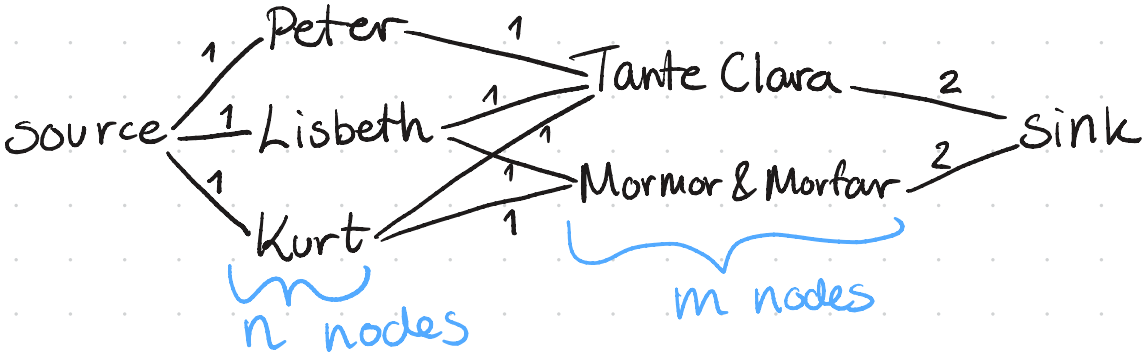
start by adding a source and sink.

Add a node for each of the children and connect them to the source with an edge of weight 1

Add a node for each host and connect them to the sink with an edge that has the capacity of the host as weight

connect children and hosts according to possible matches with weight 1

For the example instance:



The max flow of this graph is the maximum matching of children and hosts

c) Running Time

???

6

a) Which is NP-Complete

Stacks (P_1)

b) Which other NP-problem do we consider?

Subset sums

c) We need to prove...

$P_2 \leq_p P_1$, i.e. the known NP-problem can be reduced to our NP-problem

d) NP reduction

Claim: Stacks is NP-Hard.

Proof: We will reduce from subset sums, which is known to be NP-hard.

Let W and $\{w_1, \dots, w_n\}$ be an instance of subset sums, for which $W = \frac{1}{2}\text{sum}(\{w_1, \dots, w_n\})$

Construct an instance $\{c_1, \dots, c_m\}$ (C) of stacks as follows:

Take each w_i and add to C

For example if $S = \{1, 3, 2\}$ and $W = 3$ is an instance of subset sums then the resulting instance of stacks is $C = \{1, 2, 3\}$

(we note that the size of C is equal to the size of S)

Now let $I = \{1, 3\}$ (indices) be a solution to stacks for the instance C, then the same subset is a solution to subset sums. Conversely, if $O = \{1, 2\}$ is a solution to subset sums, then the same set is a solution to stacks

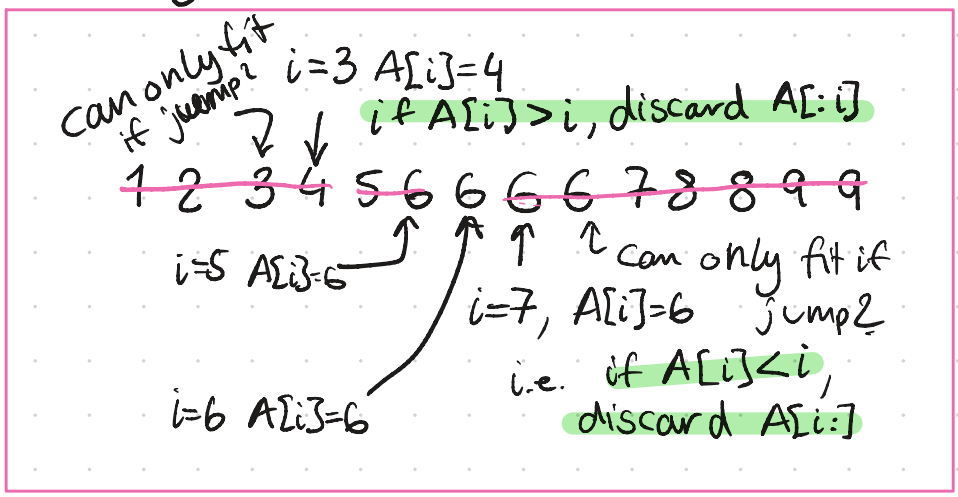
Claim: *Some Problem* is NP-hard.

Proof. We will reduce from *Other Problem*, which is known to be NP-hard.
Let ... be an instance of *Other Problem*. (...)
Construct and instance ... of *Some Problem* as follows. (...).
For example, if ... is an instance to *Other Problem* then the resuling instance to *Some Problem* is ....
(Note that the size of ... is polynomial in the size of ....)
Now let ... be a solution to *Some Problem* for the instance ....
Conversely, if ... is a solution to *Other Problem*, then ....

Exam SALD1 17 Oct 2013

1 a) Which is Divide-n-conquer Fixpoint?

b) Describe algorithm Binary Search?



(1) Identify the middle element and read the value

(2) Due to the weak staircase behavior, you may use the following roles:

- (a) A[i]==i Return i
- (b) A[i]<i Search left half
- (c) A[i]>i Search right half
- (d) Empty (no items left) Return -1

c) Running Time

log(n)

2 a) Which one is graph traversal?

Compression → MST on connected comps.

b) Describe algorithm

Given a graph G with k connected components we apply Prim's algorithm to find minimum spanning trees on each component, while keeping track of discarded edges.

This will result in a graph with the same connected components but the smallest amount of edges. We print the discarded edges

c) State the running time

Prims algorithm bounds the running time, as all other operations have the same or a quicker time complexity.

Prim's has time complexity O(???)

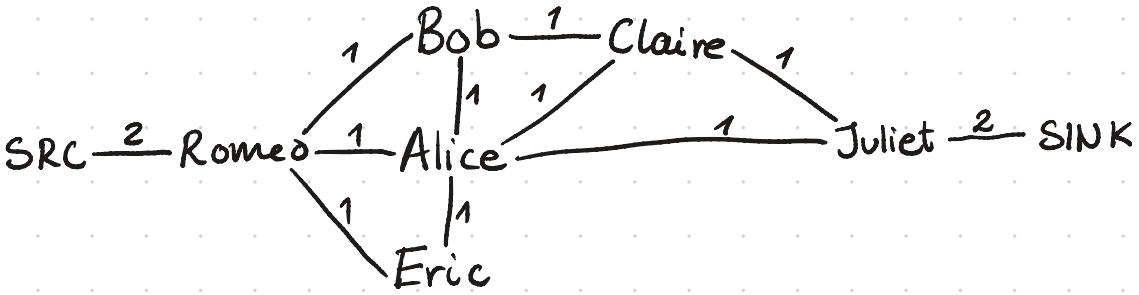
4 a) Which one is network flow?

Love Letter

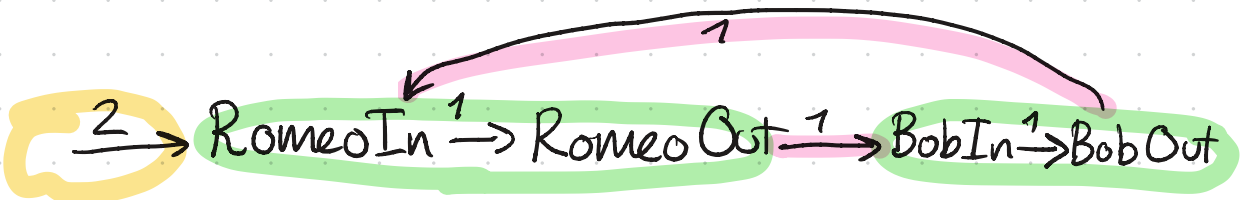
b) Describe reduction

Given facebook graph G, assign capacity 1 to all edges. Now, add a source and link it to Romeo with an edge of capacity 2. Also, add a sink and link it to Juliet with an edge of capacity 2.

In particular for the example input our graph would look like this:



Due to the constraint of each node only being allowed to be visited once, each edge is replaced by it's directed counter and each node is split into in-node and out-node with a directed edge of capacity 1:



Max flow in this graph is the maximum paths between Romeo & Juliet → Max flow of 2 indicates the letter may be sent, anything else means impossible

c) Running Time

Ford Fulkerson is $O(|V| \cdot |E| \cdot W)$ which is $O(n^2)$

6

a) Which is NP-Hard?

Dinner P_1 (assuming that each table must form a simple cycle)

b) Which other similar

Hamiltonian Cycle P_2

c) which direction?

We want to prove $P_2 \leq_p P_1$

The template is here given for a reduction that shows that *Some Problem* (presumably a new problem, maybe one in an exercise or exam) is NP-hard by a reduction from *Other Problem* (presumably a well-known problem whose NP-hardness is already established.) In other words, we want to prove *Other Problem* $\leq_p$ *Some Problem*.

Claim: *Some Problem* is NP-hard.

Proof. We will reduce from *Other Problem*, which is known to be NP-hard.
Let ... be an instance of *Other Problem*. (...)
Construct and instance ... of *Some Problem* as follows. (...).
For example, if ... is an instance to *Other Problem* then the resulting instance to *Some Problem* is ....
(Note that the size of ... is polynomial in the size of ....)
Now let ... be a solution to *Some Problem* for the instance ....
Conversely, if ... is a solution to *Other Problem*, then ....

d) describe Reduction

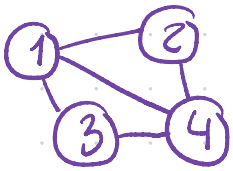
Claim: Dinner is NP-Hard

Proof: We will reduce from Hamiltonian cycle which is known to be NP-Hard.

Let the undirected connected graph G be an instance of Hamiltonian cycle. Let n denote the total number of nodes in G. Construct an instance of Dinner as follows:

Use graph G, unchanged, as facebook Graph
Let $k=1$ and $r_1 = n$ (i.e. One table with all attendees)

For example if below is an instance of Hamiltonian cycle



then the resulting instance of Dinner is the same graph and $k=1$ & $r_1 = 4$

Note that the sizes of these instances are (both) $O(n+|E|)$ (edge cnt)

Now Let $s = 1, 2, 4, 3$ be a solution to Dinner for the instance, then S constitutes a Hamiltonian cycle. Conversely, if $S_H = 1,3,4,2$ is a solution to HC, it constitutes a solution for Dinner.

Exam SALD 1

27. Febo 2014

1 a) which is Divide-n-Conquer?

Zero? (i put this as greedy though but may as well be divide-n-conquer)

2

a) which is greedy?

Zero

Taxi is just interval scheduling

b) Describe Algorithm

```
if A[1] != 0:
    return 0 // due to product, root must be zero if any leaf is zero
Start with i=1
Until reaching leaf, do:
    Get item 2·i and 2·i+1 from A
    Update i = child with v_j=0. If both are zero, the choice is arbitrary
return i
```

c) Running Time

The running time is bounded by the loop which runs in $O(\log(n))$ time

All other bits run in constant time, meaning the algorithm runs in $O(\log n)$

3

a) Which is Dynamic Programming?

Words

b) Define OPT

For each position in the string, starting at position 1, we evaluate whether S[i:j] is a valid word. Note that we have i = 1 / letter after previous word & j = current pos.

Now we can state 3 cases:

$$OPT(i,j)\begin{cases} \text{if } j > n \rightarrow \text{None} & \leftarrow \text{exceeded string} \\ \text{if } f(S[i:j]) == \text{False} \rightarrow OPT(i, j+1) & \leftarrow \text{not a word} \\ \text{Otherwise} \begin{cases} [S[i:j] + OPT(j+1, j+2)] & \leftarrow \text{take word found} \\ OPT(i, j+1) & \leftarrow \text{continue searching} \end{cases} \end{cases}$$

Not-None of: Combine into None if OPT=None

c) Running Time & Space

Time: $O(n^2)$

Space: $O(n^2)$

4

a) Which is network flow?

Drink

b) Describe the reduction

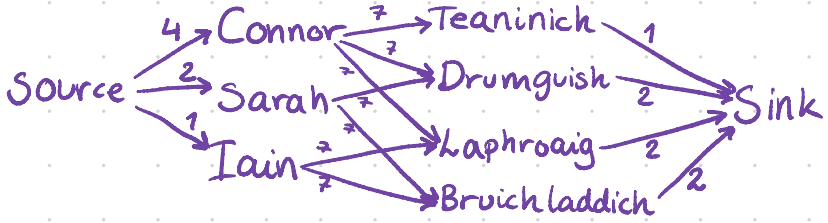
We start by initializing the source & sink nodes.

We add a node for each of the n members and add a directed edge from source → member with capacity corresponding to their drinkable-glasses-count.

We add a node for each of the m whiskeys and add a directed edge from whiskey → sink with capacity corresponding to the number of glasses left.

Lastly, we add directed edges from member → whiskey corresponding to their preference list. We give each of these edges capacity of total number of glasses left. (or inf)

For the example instance, this would yield:



Max Flow in this graph is equal to the maximum number of glasses that can be drunk by the members. If the flow is equal to the total number of glasses, then the solution is "yes" and otherwise "no".

The size of the network is $|V| = n + m + 2$

$|E| = n + m + O(n \cdot m)$ (all members like all whiskeys)

c) Running Time

Ford-Fulkerson is $O(|E| \cdot C)$ which in our case is $O(n \cdot m \cdot C)$ where C is the total number of whiskey glasses. $C \geq m$, $O(n \cdot C^2)$

6

a) which is NP-Complete?

Tables (find Complete subgraph of size r_i)

b) which other NP-Hard Problem?

Set Cover

The template is here given for a reduction that shows that *Some Problem* (presumably a new problem, maybe one in an exercise or exam) is NP-hard by a reduction from *Other Problem* (presumably a well-known problem whose NP-hardness is already established.) In other words, we want to prove *Other Problem* $\leq_p$ *Some Problem*.

Claim: *Some Problem* is NP-hard.

Proof. We will reduce from *Other Problem*, which is known to be NP-hard.
Let ... be an instance of *Other Problem*. (...)
Construct and instance ... of *Some Problem* as follows. (...).
For example, if ... is an instance to *Other Problem* then the resuling instance to *Some Problem* is ....
(Note that the size of ... is polynomial in the size of ....)
Now let ... be a solution to *Some Problem* for the instance ....
Conversely, if ... is a solution to *Other Problem*, then ....

c) Direction of Reduction

Set Cover $\leq_p$ Tables

d) Reduction

Claim: Tables is NP-Hard

Proof: We will reduce from Set Cover, which is known to be NP-Hard.

Let U be an instance of Set Cover with subsets denoted $S_1, \ldots, S_c$ and total number of elements k.

Construct an Instance r_1, r_2, r_3 and G as follows:

Let $r_1 = k$ and $r_2 = 0$ $r_3 = 0$

Let each item in U be a node in G
For each subset S_i, add edges between all pairs in the subset.

Exam SAD1 23. Oct. 2014

1 a) Greedy → which one?

Sweden

b) Describe the algorithm

We are interested in picking the smallest parties first until picking a party would yield a majority. I.e.:

```
Parties ← Input sorted by number of seats in increasing order
Coalition_seats_max ← ½ m (m = total seats)
```

Ties settled arbitrarily

```
For P in Parties:
    if P["seats"] < Coalition_seats_max:
        Coalition_Seats_max -= P["seats"]
        Minister = P["leader"]
    else:
        break //No reason to process remaining parties as they are only getting larger
Return Minister //The latest added party's leader (i.e. biggest)
```

c) Running Time

Sorting the parties is $O(n \log n)$
Worst case is all parties but 1 run through the for loop, i.e. $O(n)$
Inner part of for-loop may run in $O(1)$

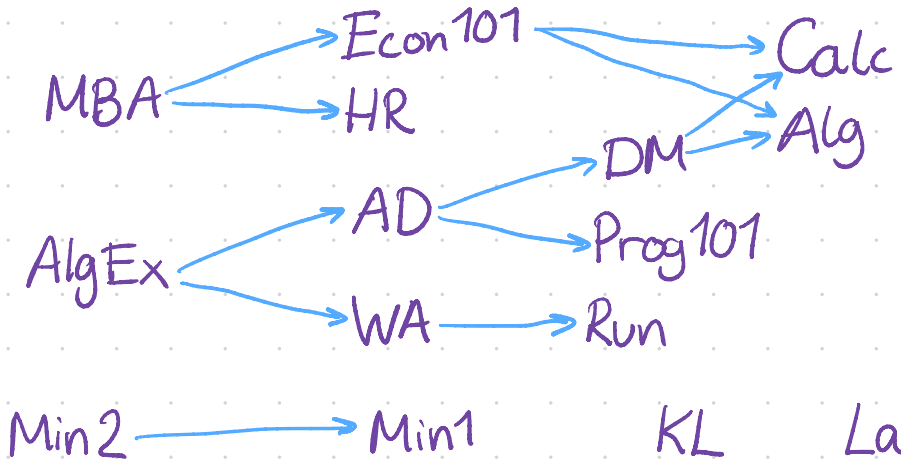
Thus the Running time of the algorithm is bounded by the sorting and is thus $O(n \log n)$

2 a) Which one is graph traversal?

Courses

b) Describe graph and algorithm

The graph of the courses is constructed as a Directed-Acyclic-Graph (DAG) with each node corresponding to a course and edges indicating dependencies from course → dependency. The example Input constructs the following graph:



To find and order the courses, find the subgraph which is AlgEx and all nodes reachable from this.

Output the subgraph in order from leafs → root such that for any node p with children $c_1, \ldots, c_k$ The children $c_1, \ldots, c_k$ occur before p in the list.

track nodes added and reverse the order?

c) Running Time

?? O(L)? $O(L^2)$?

3 a) Which is dynamic Programming?

Count

b) Give the reccurrence relation OPT

Given i = remaining steps and returning the number of options:

$$OPT(i) = \begin{cases} \text{if } i=1 \text{ then } 1 \\ \text{if } i=2 \text{ then } 2 \\ \text{otherwise } \underbrace{OPT(i-1)}_{\text{if we take 1 step}} + \underbrace{OPT(i-2)}_{\text{if we take 2 steps}} \end{cases}$$

c) Running Time + Space

Space is $O(n)$ as we are filling out the matrix for each i from n until reaching base case.

Filling out the matrix takes time $O(n)$ as well, and bounds the running time of our algorithm

4 a) Which is network flow?

Detour

b) Describe the reduction

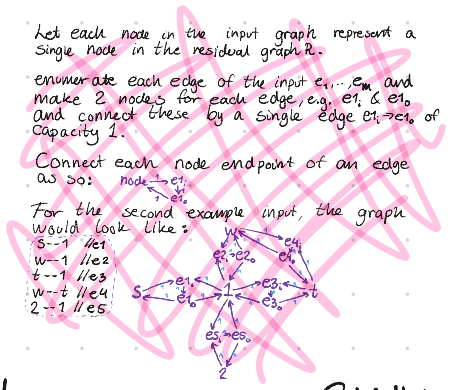
Let each node in the input graph represent a single node in the residual graph R.

enumerate each edge of the input $e_1, \dots e_m$ and make 2 nodes for each edge, e.g. $e1_i$ & $e1_o$ and connect these by a single edge $e1_i \to e1_o$ of capacity 1.

Connect each node endpoint of an edge as so: node → $e1_i$, $e1_o$

For the second example input, the graph would look like:

s--1 //e1
w--1 //e2
t--1 //e3
w--t //e4
2--1 //e5



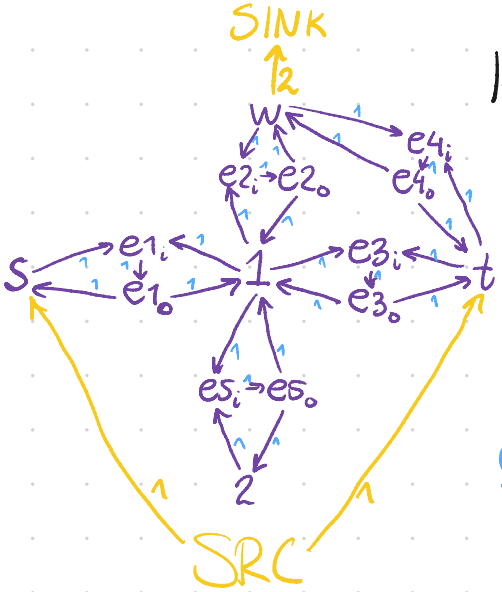
Marie's Solution:

Node disjoint path problem:

I create a super-source SRC and super sink SINK. I create two nodes for each node in the graph connected by a directed edge of capacity 1. I name the nodes $\langle name \rangle_{in}$ and $\langle name \rangle_{out}$ such that the edge is $\langle name \rangle_{in} \to \langle name \rangle_{out}$. I then add an edge from SRC to each of s & t with capacity 1 and an edge from w to SINK.

I think this is incorrect as it does not guarantee edges disjoint paths? → in our problem we are allowed to pass through a node twice if there are sufficiently many edges

I think mine + a little is better such that:



If maxflow = 2 then there exists edge disjoint paths s→w t→w which constitutes an edge disjoint path s→w→t

We may backtrack to find this solution path!

c) Running Time $O(|E| \cdot C)$ (C = 2, bounded by edge to sink)

Since C=2, Time Complexity is $O(|E|) = O(5 \cdot m) = O(m)$

5 a) Is Detour a decision problem?

No!

Decision problem version: Does there exist a a path from s to t which passes through w and uses each edge at most once?

b) Certificate

A certificate for a true instance would be the given path.

For example for the second instance, it would be true and the certificate would be s1wt (path)

Certificate size is O(m) as we may use each edge at most once in the path

c) How may certificate be checked?

Check that

(1) path contain w and has s & t as endpoints

(2) path is valid in graph

Both of these can be performed in O(m)

6

a) Which is NP-Hard?
Visitor

b) Which NP-Hard problem is similar?
Longest Path

c) Which Direction is the reduction?
Longest Path $\leq_p$ Visitor

d) Reduction

The template is here given for a reduction that shows that *Some Problem* (presumably a new problem, maybe one in an exercise or exam) is NP-hard by a reduction from *Other Problem* (presumably a well-known problem whose NP-hardness is already established.) In other words, we want to prove *Other Problem* $\leq_p$ *Some Problem*.

Claim: *Some Problem* is NP-hard.

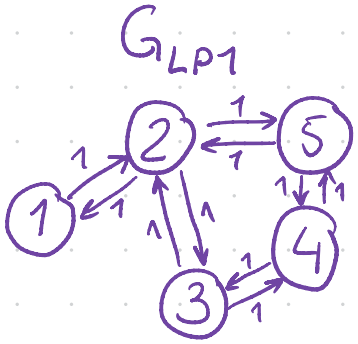
Proof. We will reduce from *Other Problem*, which is known to be NP-hard.
Let ... be an instance of *Other Problem*. (...)
Construct and instance ... of *Some Problem* as follows. (...).
For example, if ... is an instance to *Other Problem* then the resulting instance to *Some Problem* is ....
(Note that the size of ... is polynomial in the size of ....)
Now let ... be a solution to *Some Problem* for the instance ....
Conversely, if ... is a solution to *Other Problem*, then ....

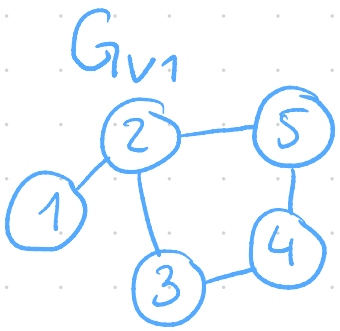
Claim: Visitor is NP-Hard

Proof: We will reduce from Longest Path, which is known to be NP-Hard.
Let the directed weighted graph G_L be an instance of Longest path, for which all edge weights are 1.

Construct an instance G_V, W of Visitor for which G_V is the undirected unweighted version of G_{LP}. Let W contain all nodes that have an incoming edge of weight 1. I.e. $W = V$.

For example if G_{LP1} is an instance of LP, then the resulting instance of Visitor is G_{V1}:





$W_1 = \{1,2,3,4,5\}$

(Note that the size of both graphs is $O(m+n)$
Now let $k=5$ be a solution to Visitor for the the instance, then $L=5$ is a solution to Longest path.
Conversely a solution to LP constitutes a solution to Visitor of the same value

Exam SALD1 26. Feb 2015

1 a) Which one is graph traversal?

Avoid

b) Explain Solution

Take the input graph G and remove all Red vertices. (Or simply disregard when doing construction)

Then, find the connected component using BFS of S (Reachability tree) and backtrace the path to t if t is reachable from s.

c) Running Time

Construction is the same, Recon is $O(m+n)$

BFS takes $O(m+n)$ and backtracking $O(n)$

The overall Runtime is therefore $O(m+n)$

2 a) Which is Divide-n-conquer?

Hole

b) Describe solution and base case

Since we assume n is a power of 2, we may divide the board in quarters as small as 2x2 squares.

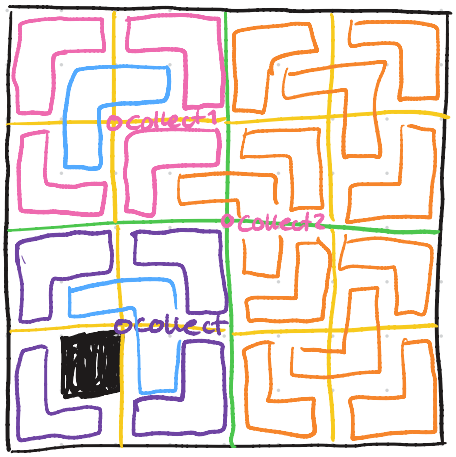
We may also initially determine that some n simply don't have a solution if $(n \cdot n-1) \% 3 != 0$

squares should be divisible by 3

When we have quartered all the way to 2x2 squares we may consider 2 cases:

(1) square has the red square
(2) square is unfilled

In case (1) we place the L to fill out remaing squares. In case 2 we place the L such that exactly 3 empty squares are in the "collection point" and a possible 4th empty square is pointing toward next "collection point" I.e. for 2x8:



Split 1
Split 2
placement of 3
placement of collect
* placement of 3+1
* placement of collect
repeat and final collect

c) Running Time

We divide into $\log_2(n)$ squares, each constant time processing

$O(\log(n))$? $\Theta(n \log n)$?

3 a) Which one is DP?

More DAG

b) Define recurrence relation OPT

We use DP to traverse the DAG from $s \to t$.
Given node n, we define:

$$OPT(n) \begin{cases} 0 \text{ if } n \text{ is } t & \text{// } t \text{ is never red and starts returning} \\ \text{Null if } n \text{ is childless} & \text{// } t \text{ not found and can't progress} \\ \underbrace{\text{Max\_not\_null}([OPT(c_i) \text{ for } c_i \text{ in child of } n])}_{\text{Return null if all are null, else the max. not-null val}} + 1 \text{ if } n \subseteq W & \text{// add to red-count} \end{cases}$$

When running OPT(s) we will traverse the DAG and determine the maximum #red nodes in a path to t from a given node n.

c) Running Time & Space

Memory matrix is size $O(n)$ and running time is $O(n)$ for filling out the matrix.

4 a) Which is flow?

Forced

b) Describe the reduction

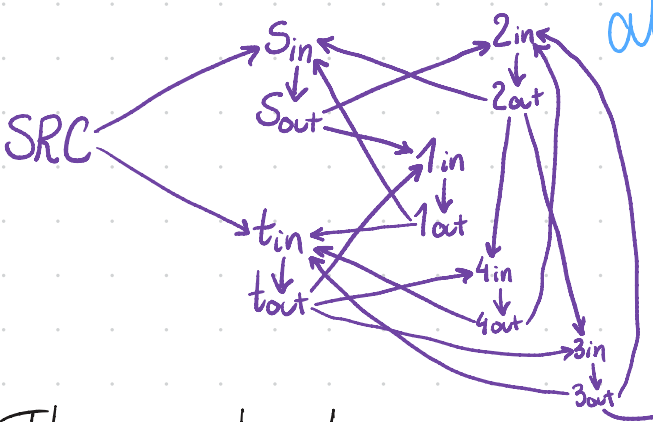
We want to find 2 node disjoint paths s-r and r-t. We do this by constructing a directed weighted graph as follows:

Create 2 nodes for each $v \in V$ named v_{in} and v_{out}. Add an edge of capacity 1 $v_{in} \to v_{out}$.

For each edge $e \in E$ add 2 directed of capacity 1 linking $e_{v_1 out} \to e_{v_2 in}$ and $e_{v_2 out} \to e_{v_1 in}$.

Lastly add source SRC and sink SINK. Add 2 directed edges from SRC $\to s_{in}$ and SRC $\to t_{in}$ with capacity 1. Add a directed edge from the red node $\to$ SINK with capacity 1.

For the example input this would look like:



all weights are 1

The maximum flow in this graph is the number of disjoint paths. If $k=2$ one may backtrack flow to SINK determine paths.

The graph has size $|V| = 2 \cdot n + 2$ $\quad |E| = 2 \cdot m + 3$

c) Running time

Using Ford-Fulkerson (p. 344 of textbook) the running time is $O(|E| \cdot C) = O((2m+3) \cdot 2) = O(m)$

5

a) is Avoid a decision problem?

No, decision problem version:

Does there exist a path from s to t avoiding red nodes?

b) Describe certificate

The certificate for a "yes"-instance is the path from s to t avoiding red nodes.
The certificate is size $O(n)$

c) How may the certificate be checked?

Going through the path and checking that
(1) There exists an edge in G for each step in the path
(2) None of the nodes in path are red

Running time of this is $O(n^2)$, bounded by the adjacency check in a complete graph.

6

a) Which is NP-Hard?

More

b) Which NP-Hard problem is similar?

Longest Path

The template is here given for a reduction that shows that *Some Problem* (presumably a new problem, maybe one in an exercise or exam) is NP-hard by a reduction from *Other Problem* (presumably a well-known problem whose NP-hardness is already established.) In other words, we want to prove *Other Problem* $\leq_p$ *Some Problem*.

Claim: *Some Problem* is NP-hard.

Proof. We will reduce from *Other Problem*, which is known to be NP-hard.
Let ... be an instance of *Other Problem*. (...)
Construct and instance ... of *Some Problem* as follows. (...).
For example, if ... is an instance to *Other Problem* then the resuling instance to *Some Problem* is ....
(Note that the size of ... is polynomial in the size of ....)
Now let ... be a solution to *Some Problem* for the instance ....
Conversely, if ... is a solution to *Other Problem*, then ....

c) Direction

Longest Path $\leq_p$ More

c) Describe Reduction

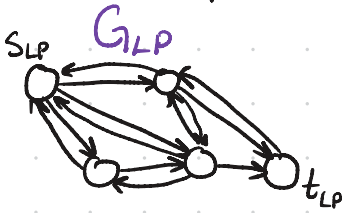
Claim: More is NP-Hard

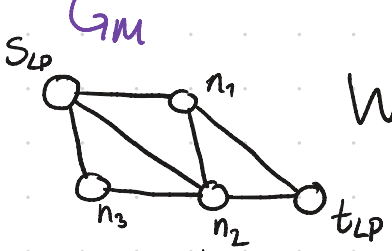
Proof: We will reduce from Longest Path, which is known to be NP-Hard.

Let G_{LP}, s_{LP}, t_{LP} be an instance of Longest Path for which G_{LP} is a directed graph with weight 1 on all edges and $\{s_{LP}, t_{LP}\}$ being in the vertex set of G_{LP}.

Construct an instance G_M, s_M, t_M & W_M of More as follows:
Let $s_M = s_{LP}$, $t_M = t_{LP}$ and G_M be the undirected unweighted version of G_{LP}.
Let W_M contain all vertices whose incoming edges have weight 1. In particular
Let $W_M = V_{LP}$ (the entire vertex set of original graph)

For example the construction may look as follows:





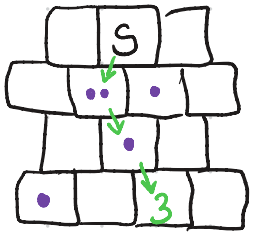
$W = \{s_{LP}, t_{LP}, n_1, n_2, n_3\}$

Note that the size of both instances is $O(|V|+|E|)$

Now let path S_M be a solution to More, then the same path constitutes a solution to the longest path problem Conversely a solution S_{LP} to Longest Path constitutes a solution to More.

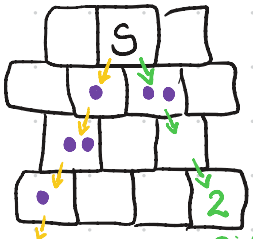
Exam SAD1 5. Januvary 2018

1 a) Give an example where greedy works



Picks highest immediate reward and gets highest overall reward

b) Give a counter example



4 optimal Picked by greedy

c) Running Time

At each iteration 2 values are fetched and compared, which may be done in constant time.

We do O(n) such iterations yielding total runtime complexity of O(n)

2 a) Which is graph traversal?

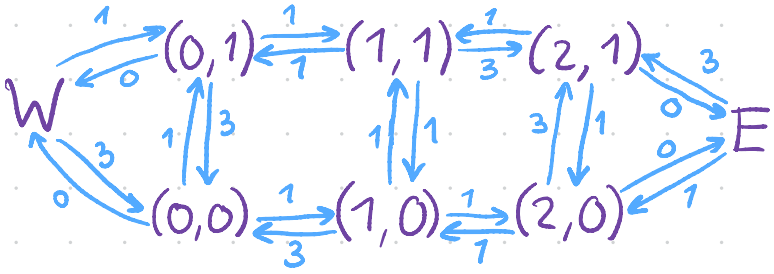
East-West Relations

b) Explain how to model the graph problem

Each grid position (except N, S & .) represent a node in graph G. E-nodes are unioned into one singular node and so are W-nodes.

Nodes are connected by directed weigted edges indicating all possible moves from a given square and given the weight/cost of the target square.

I.e. for the second input we construct graph:



c) Which algorithm?

We want to find the shortest simple path in G. We use Djikstras (p. 180 of textbook) to find the shortest path from E to W.

d) Running Time

With Heap Priority Queue, Djikstra's runs in $O(|E| \cdot \log |V|)$

We denote the number of squares in the input n, then G has $|V| = O(n)$ and $|E| = 4 \cdot O(n) = O(n)$

Giving us final runtime complexity $O(n \log n)$

3 a) Which is dynamic programming?

Clean The Wall

b) Define the recurrence relation OPT

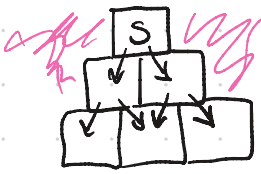
Let i denote the row numbers $\{i, \dots, N\}$
Let j denote the brick numbers from left→right
Iterate from $i = N$, $j = N/2$ and find optimal traversal from brick (i,j) using:

$$OPT(i,j) = \begin{cases} c(i,j) \text{ if } i = 1 \quad \text{// Bottom row, start returning} \\ \text{when } i\%2 = 0 \\ \quad \max(\underbrace{OPT(i-1,j)}_{\text{Left}}, \underbrace{OPT(i-1,j+1)}_{\text{Right}}) + \underbrace{c(i,j)}_{\text{brick reward}} \\ \text{else} \\ \quad \max(OPT(i-1,j-1), OPT(i-1,j)) + c(i,j) \end{cases}$$

c) Running Time & Space

The Space for the matrix is $\frac{N^2 + N(N-1)}{2}$ (almost N^2)

Yet we may note something interesting about the Reachable nodes for the traversal pattern. The reachable nodes always form a pyramid structure:



anything in these spots is not reachable

The brick count we iterate over is therefore
$N + N-1 + N-2 + \dots + 1 = \frac{N(N-1)}{2}$

No matter what, both Space (matrix) and Running time (calculating matrix) is $O(N^2)$

4 a) which is flow?

Ambassadors II

b) Describe the reduction

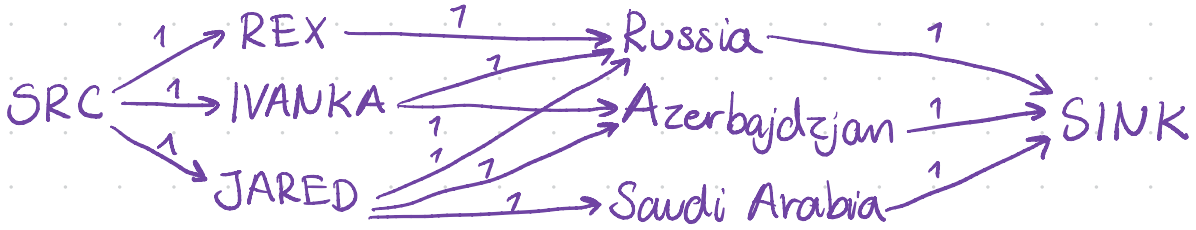
We start by constructing a source node SRC and sink node SINK.

We then create M nodes representing the countries. We add an edge from each country node to the SINK, with capacity 1.

We create N nodes representing the candidates. We add an edge from SRC to each candidate i with capacity corresponding to $c(i)$, the maximum number of countries the given candidate can be an ambassador for.

At last, we add an edge of capacity 1 for each business relation, linking from candidate → country.

For example instance 1 the graph would look like:



The max flow of this instance corresponds to the maximal number of countries matched with an ambassador.

The size of the constructed graph is $|V| = N+M+2$

$|E| = N+M+O(N \cdot M)$

c) Running Time

Using Ford-Fulkerson (p. 344 of textbook) the runtime is $O(|E| \cdot c)$

$C = M$ in our case giving complexity $O((N+M+MN)M) = O(M^2N)$

5 a) Which is NP-Hard?

Russian Interference

b) Which other NP-Hard problem is similar?

Vertex Cover

The template is here given for a reduction that shows that *Some Problem* (presumably a new problem, maybe one in an exercise or exam) is NP-hard by a reduction from *Other Problem* (presumably a well-known problem whose NP-hardness is already established.) In other words, we want to prove *Other Problem* $\leq_p$ *Some Problem*.

Claim: *Some Problem* is NP-hard.

Proof. We will reduce from *Other Problem*, which is known to be NP-hard.
Let ... be an instance of *Other Problem*. (...)
Construct and instance ... of *Some Problem* as follows. (...).
For example, if ... is an instance to *Other Problem* then the resuling instance to *Some Problem* is ....
(Note that the size of ... is polynomial in the size of ....)
Now let ... be a solution to *Some Problem* for the instance ....
Conversely, if ... is a solution to *Other Problem*, then ....

c) Direction

Vertex Cover $\leq_p$ Russian I

d) Reduction

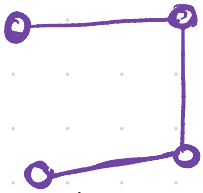
Claim: Russian Interference is NP-Hard

Proof: We will reduce from Vertex Cover, which is known to be NP-Hard.
Let the undirected unweighted graph G be an instance of Vertex Cover with the goal of finding the minimum k.

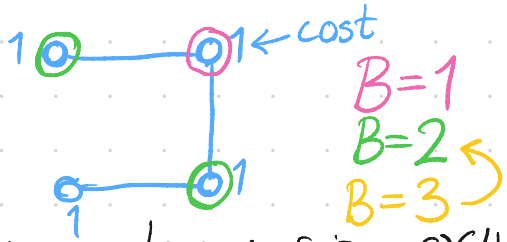
Construct an instance of Russian interference as follows:

Use the same graph G and associate a cost $c(i)=1$ to all nodes. Now, Let the budget $B=k$. For example:

Vertex Cover



Russian Interference



(Note that both instances have size $O(|V|+|E|)$)
Now Let $S=2$ be a solution to Russian Interference for the line graph instance, then $k=2$ is a minimal Vertex Cover.
Conversely, a minimal solution k to Vertex Cover constitutes a maximum budget S in Russian Interference

Exam Algorithm Design 7 januari 2019

1 a) which is greedy?

Some Times

b) Describe Algorithm

Picking k non-negative numbers giving the maximal product is equivalent to picking the k largest numbers ≥ 1

In pseudo-code one might write:

```
Sort Input in decr. order
If Input[0] < 1 : // if greatest item < 1
    return Input[0] // it will not pay off to multiply
                       fractions together
Result = 1
For i in range(k):
    if Input[i] > 1:
        Result *= Input[i]
    else:
        Return Result // if remaing is ≤ 1, no higher product
Return Result
```

c) Running Time

For-loop is $O(k)$, Sorting is $O(n \log n)$
everything else may be done in constant time
Overall Complexity is therefore $O(n \log n)$

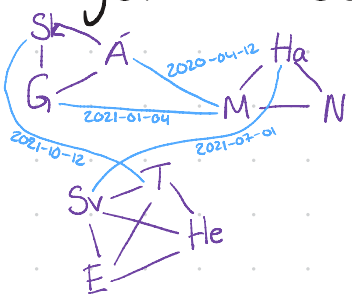
2 a) Which is graph?

Faroe Islands

b) How is it modelled as a graph problem? +Ex 1

We may consider each village V_i to be a node in the graph. We then connect all village nodes in the same community C_j to each other using undirected edges.

At last we add undirected edges annotated with the date for each bridge/tunnel between the cities it connects.



For example 1:

c) Describe algorithm

I will use BFS to determine Reachability (i.e. connected component) From Sk. → T

In particular I will start with the graph without any bridges/tunnels and determine the connected component of Sk.

Then i add each edge in the order of building determing whether T has become reachable in the given step. We may note that we only continue traversing if one end is in the connected comp of Sk while the other is not.

We return the date of the build connecting the lovers or "heartbreak" if never connected

d) Running Time

BFS Runs in $O(N^2+M)$

3 a) which one is Dynamic Programming?

Grave Robber

b) Define the recurrence relation OPT

$$OPT(i) \begin{cases} \text{if } i = N \text{ then } g_i \text{ //Reached bottom room} \\ \text{if } D(i) \text{ is empty then None //Cannot reach bottom} \\ \text{otherwise} \\ \quad \underbrace{\text{Max\_not\_none}}_{\text{if any not-None paths exist, return max value; if all None, return None}}([OPT(j) \text{ for } j \text{ in } D(i)]) + \underbrace{g_i}_{\text{gold in room } i} \end{cases}$$

Where $i \in \{1, \ldots, N\}$ is the room number, $D(i)$ denotes the set of rooms downstairs of i and g_i denotes the gold in room i.

c) Running Time and Space

The Space of the memory matrix is $O(N)$

The Running time is the time it takes to fill out the memory matrix which is also $O(N)$

4 a) Which is flow?

Messy Arithmetic

b) Explain the reduction

We start by constructing SRC & SINK nodes, representing our source and sink in the max flow graph.

We add a node for each pair of numbers (a_i, b_i) and an edge from SRC → each pair with capacity of 1.

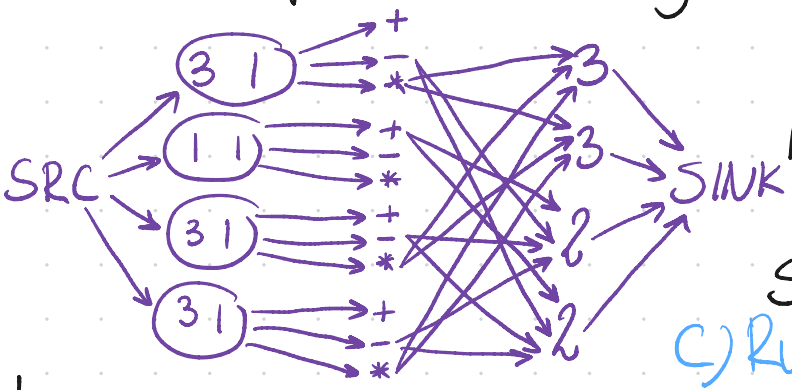
We add 3 nodes for each pair of numbers (a_i, b_i) for each of the possible arithmetic ops.

We add a directed edge with capacity 1 from the pair node to each of the corresponding arith-nodes.

We add a node for each result and an edge with capacity 1 from each result node → SINK

At last, we connect arithmetic nodes to the result that they represent, if it exists, by a directed edge with capacity 1.

For example 2 this may look like:



max flow solution graph may be backtracked to recover the arithmetics

Size is $|V| = O(n)$ $|E| = O(n^2)$

c) Running Time $O(|E| \cdot C)$

($|E| = n^2$, $C = n$)

Using Ford-Fulkerson (p.344 of textbook) Running time is $O(n^3)$