Lecture 27

Final review and epilogue

Chinmay Nirkhe | CSE 421 Winter 2026

What all have we learned?

Primarily.. a toolkit for algorithm design:

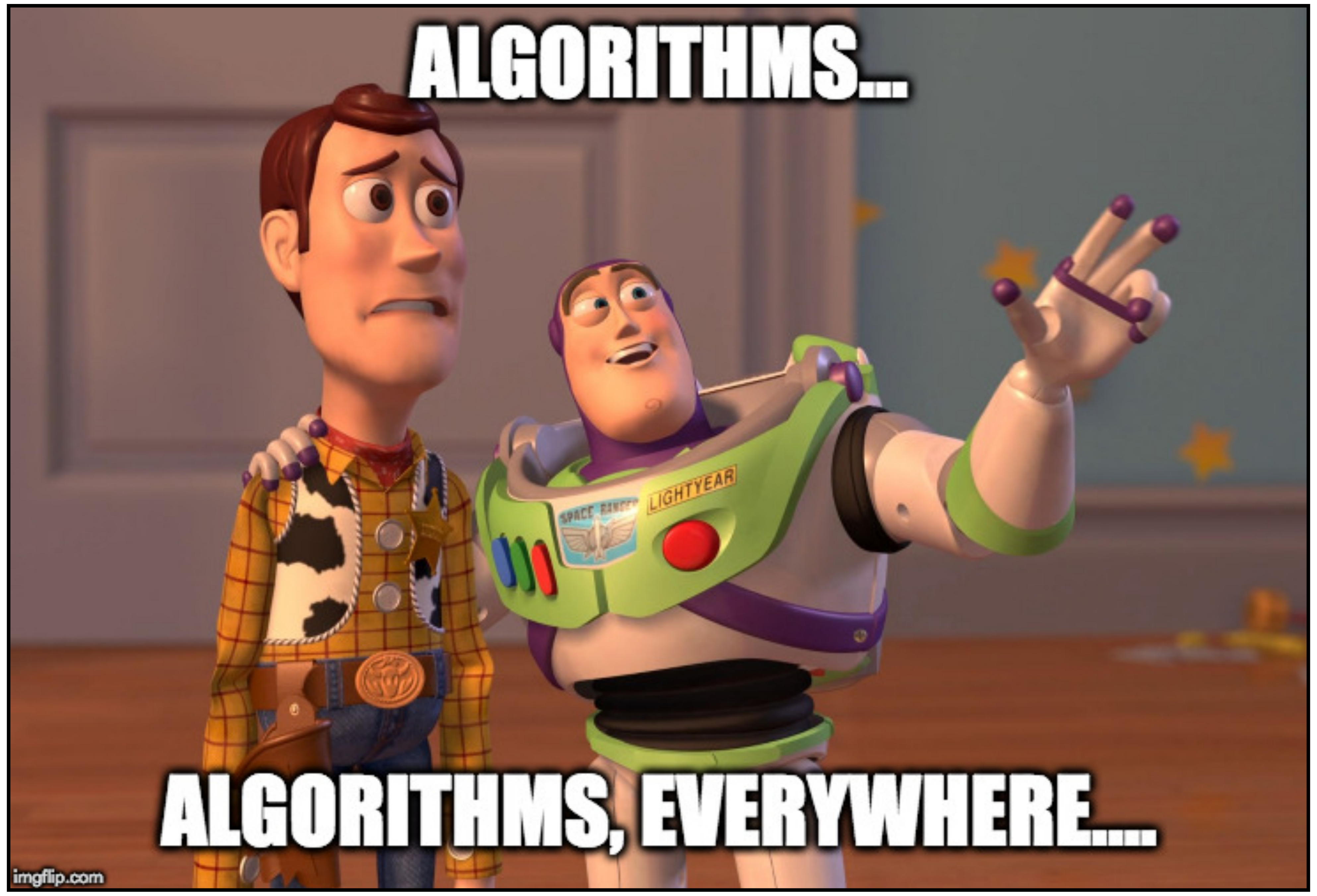
- Stable matchings
- Graph search (BFS/DFS, graph modeling)
- Greedy algorithms
- Divide and Conquer
- Dynamic Programming
- Network flow (Max Flow-Min Cut duality)
- P and NP (NP-completeness, reductions, showing a problem is NP-hard)
- Linear Programming

And now …

A recap of all we have learned

My goals for you in this course

- Help you learn to identify algorithmic problems
- Develop a toolkit for finding efficient algorithms
- Develop techniques for proving *correctness* and *analyzing* their properties
- Communicate your algorithms and their properties to others
- **New!** Learn to use AI as a tool for developing algorithms

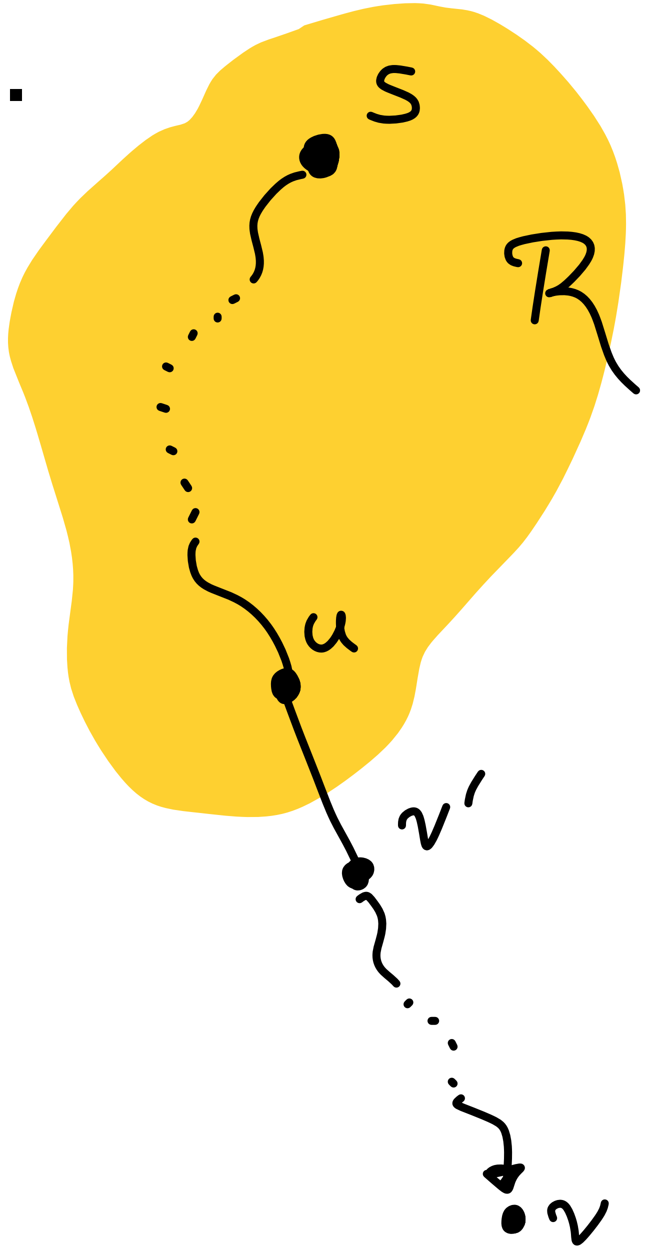


Generic graph traversa

Reachable(s): L2

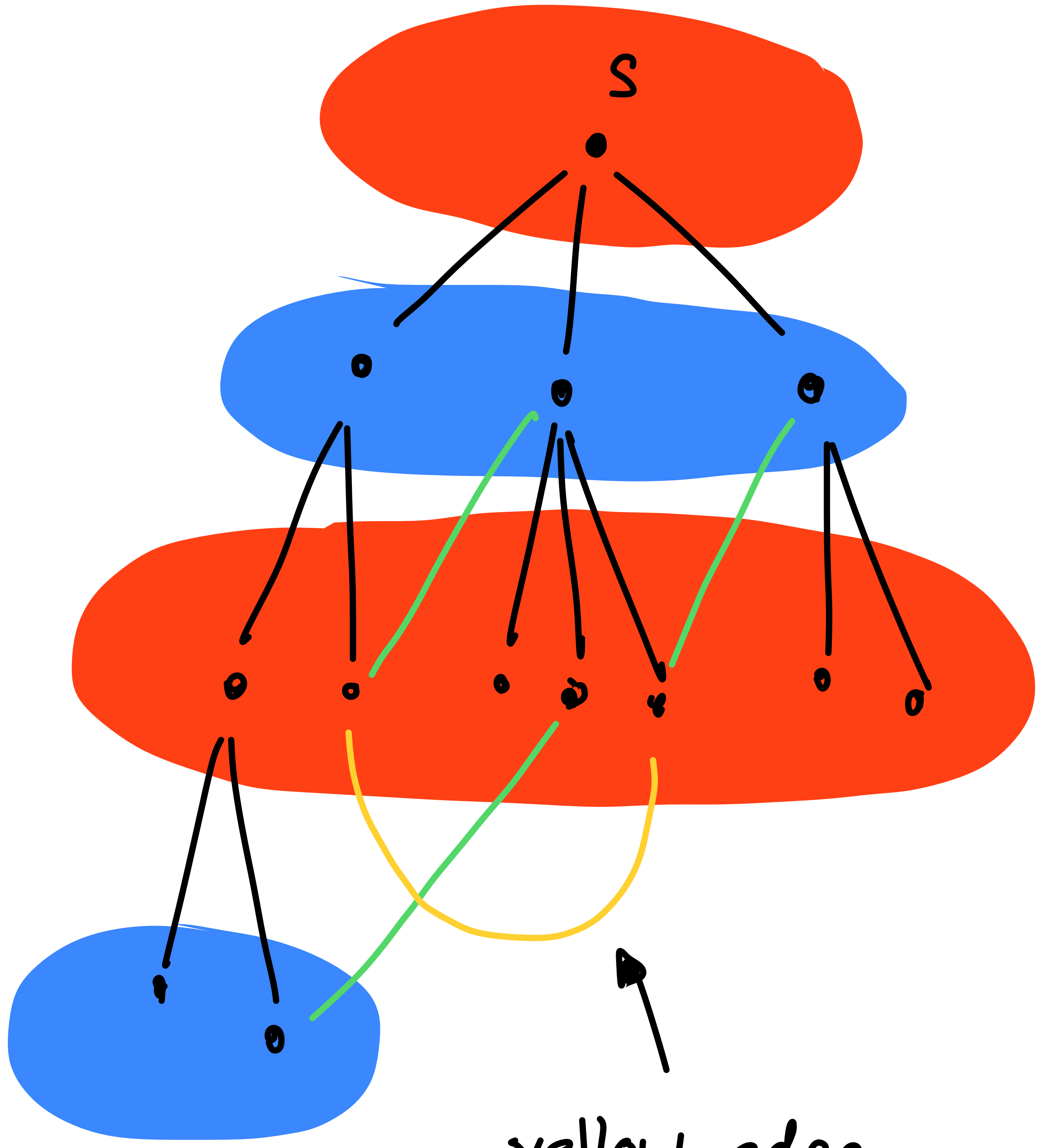
```
$R \leftarrow \{s\}$
While there exists a $(u, v) \in R \times (V \backslash R)$
    Add $v$ to $R$: $R \leftarrow R \cup \{v\}$.
return $R$
```

- **Claim:** R is exactly the set of reachable vertices.
- **Proof:** We show both directions. (1): every vertex in R is reachable. (2): every reachable is in R .
  - **Direction 1**. For $v \in R$, there is a path $s \rightsquigarrow v$. Proved by induction on the generic graph traversal algorithm: If we added v by edge $(u, v) \in R \times (V \backslash R)$ then $s \rightsquigarrow u \rightarrow v$.
  - **Direction 2**. Assume (for $\perp$), there is a vertex v that is reachable but not $v \notin R$.
    - Let $p =$ the path $s \rightsquigarrow v$ and let v' be the **first** vertex on p such that $v' \notin R$.
    - Then u, the predecessor of v', satisfies $u \in R$ and $(u, v') \in R \times (V \backslash R)$.
    - Contradicts the definition of the generic graph traversal.



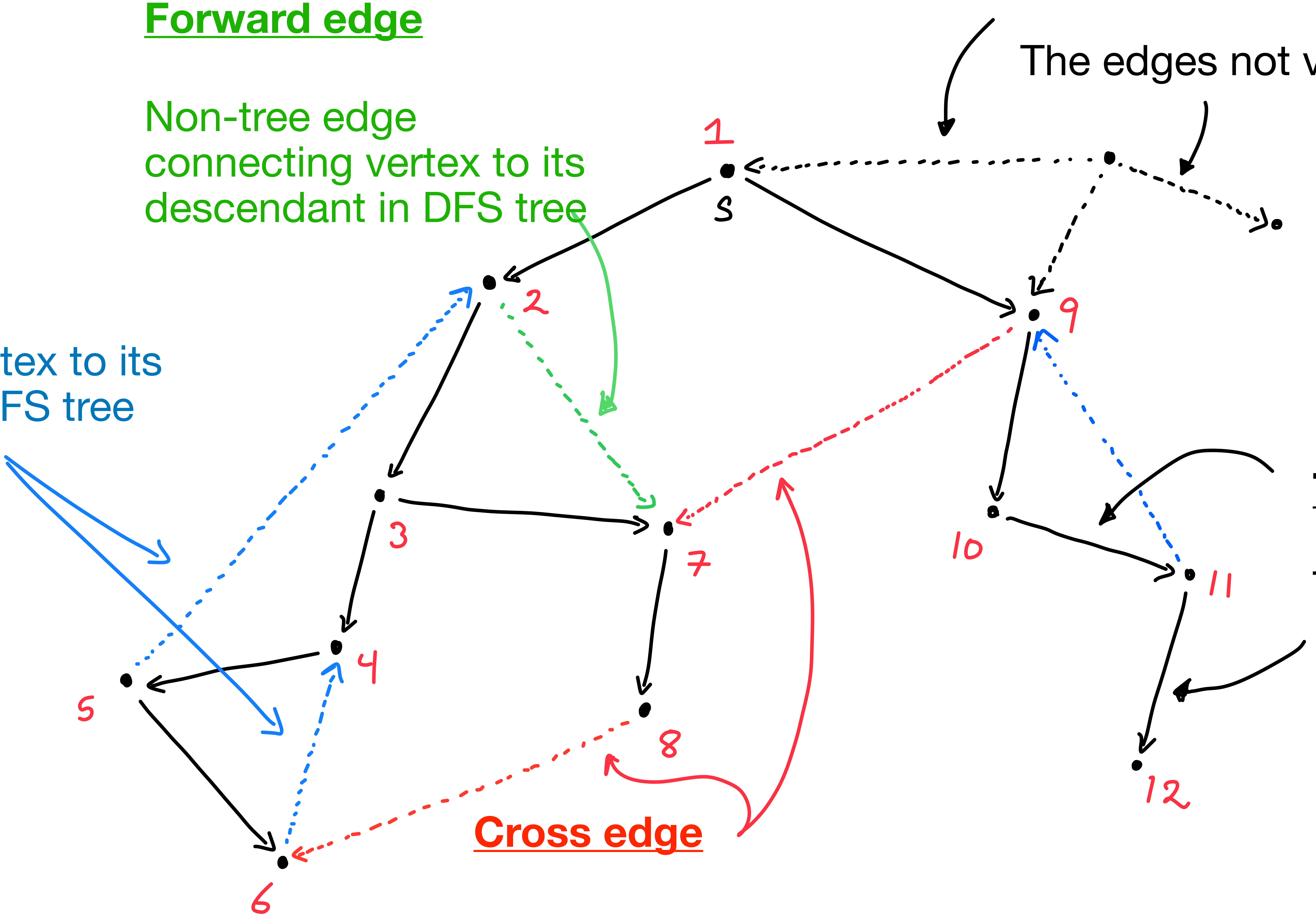
Bipartiteness testing

- **Claim:** A graph is bipartite iff it contains no *odd cycles*.
- **Algorithm:**
  - Start BFS from some vertex s. Instead of marking vertices as visited or not, marked them as "red", "blue", or "not visited". Mark s as red and add s to queue Q.
  - Pop vertex u from queue Q.
    - Check all neighbors v of u and make sure they are either "not visited" or the opposite color of u.
    - If not, abort and output "not bipartite".
    - If so, add the "not visited" neighbors v to the queue Q and color them with opposite color.
  - If queue Q is empty, output coloring generated.
- **Runtime:** Same as BFS, $O(n + m)$.



yellow edge makes this graph not bipartite

DFS edge nomenclature



Forward edge

Non-tree edge connecting vertex to its descendant in DFS tree

Unvisited edge

The edges not visited during DFS

Back edge

Connects vertex to its ancestor in DFS tree

Tree edge

The edges in the DFS tree

Cross edge

Connects vertices across branches. Always high → low in DFS tree

The principle of greedy algorithms

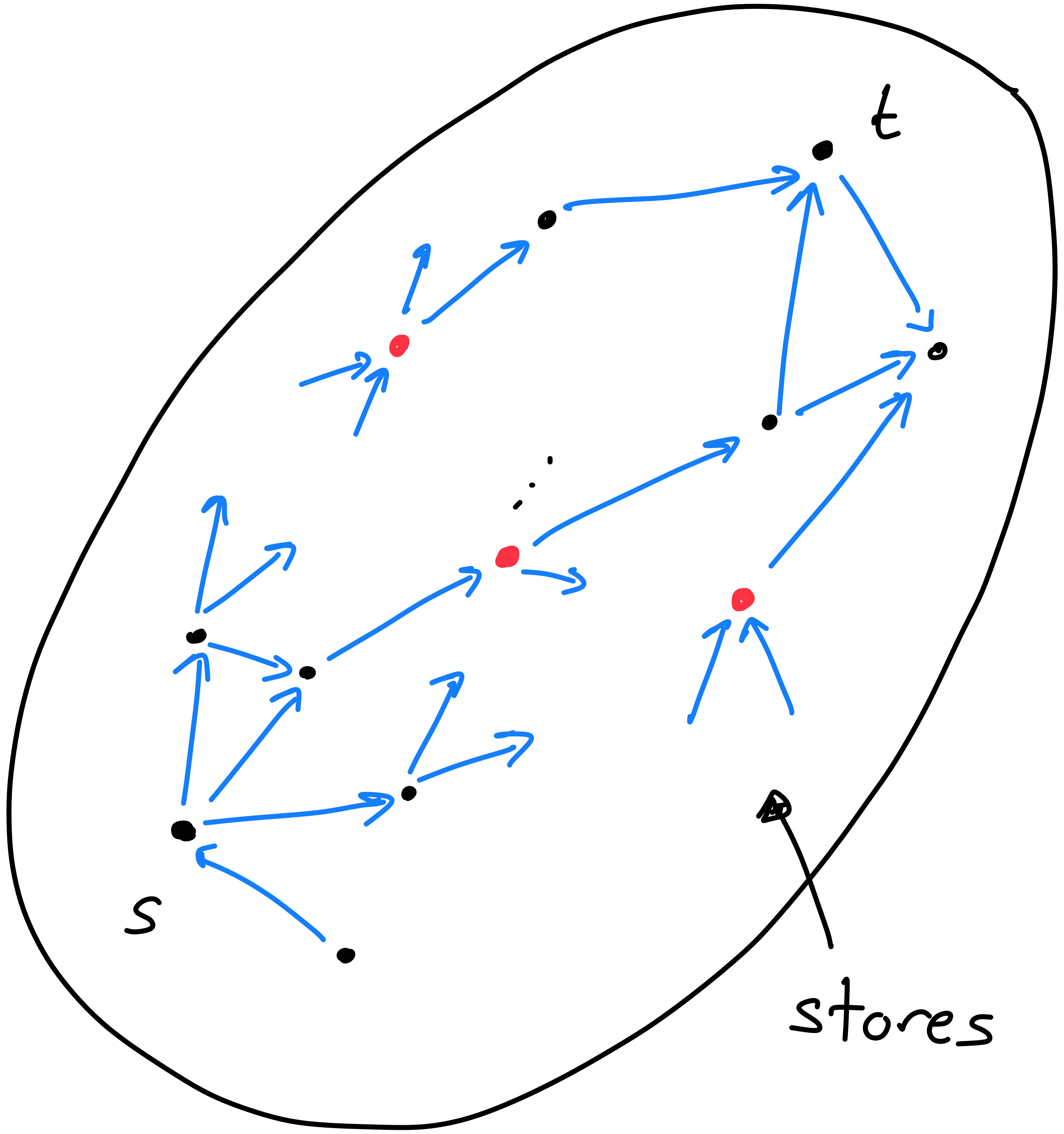
- Solving the *optimization problem* will require making many decisions (such as whether to include or not a job in the schedule)
- In a greedy algorithm, we make each decision locally without looking as to how it will effect future decisions
- Not every greedy criteria for making decisions works
    - It’s not obvious which criteria will work
    - We will focus on methods for proving that greedy algorithms do work
- When a greedy decision is made, it will be *provably* optimal

Greedy algorithm general strategy

- **Greedy algorithm stays ahead:** Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithms
- **Structural:** Discover a structure-based argument asserting that the greedy solution is at least as good as every possible solution.
- **Exchange argument:** We can gradually transform any solution into the one found by the greedy algorithm with each transform only improving or maintaining the value of the current solution.

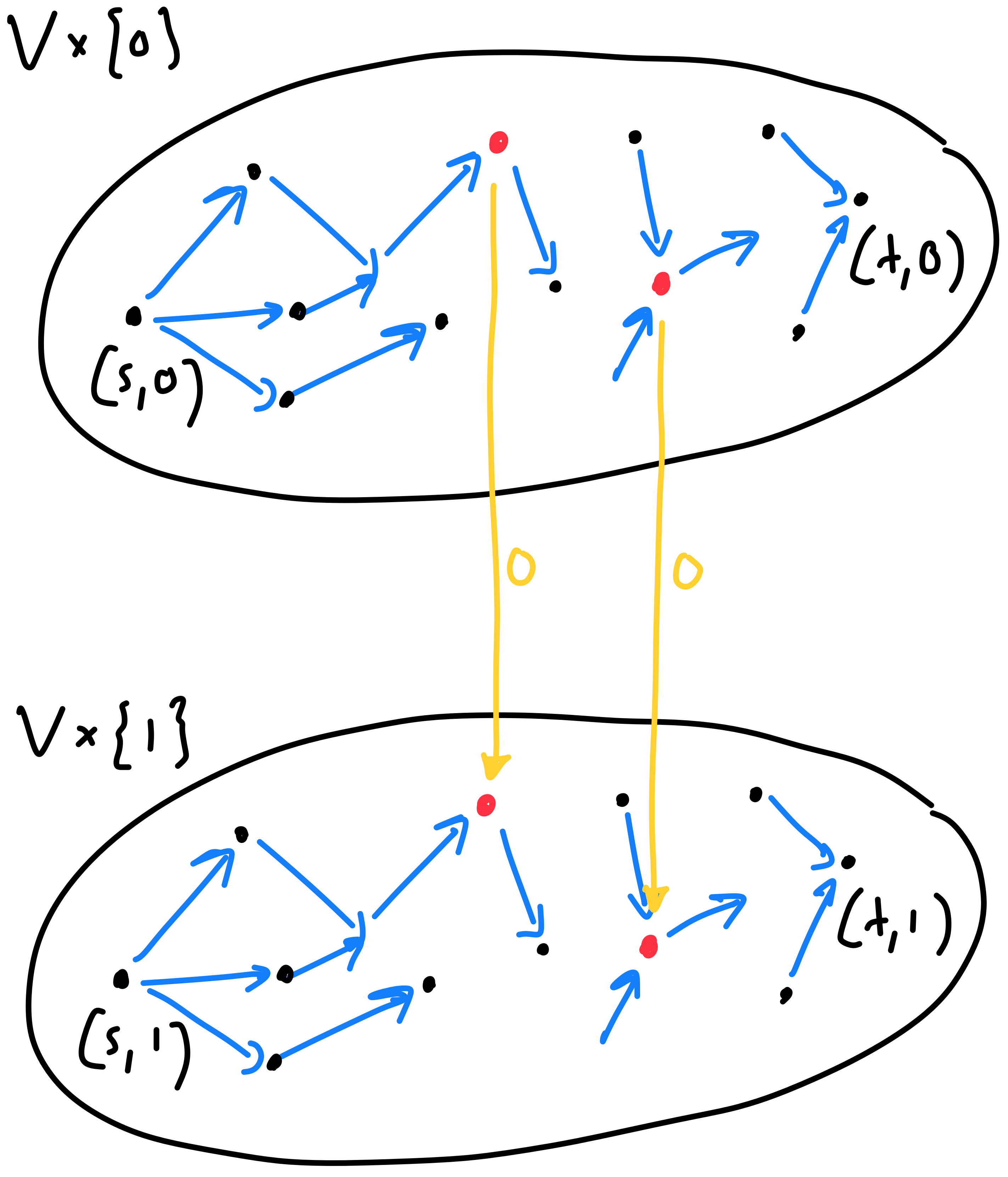
Example problem: Johnny's birthday present

- Consider a city expressed as a *directed weighted* graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}_{\geq 0}$.
- Johnny's mother starts at $s \in V$ and needs to get to the birthday party at location $t \in V$
- She forgot to buy a birthday present though and can find one at vertices $V' \subseteq V$.
- **Goal:** Calculate *the* shortest path from $s \rightsquigarrow t$ that includes a vertex of V'.



Johnny’s birthday present

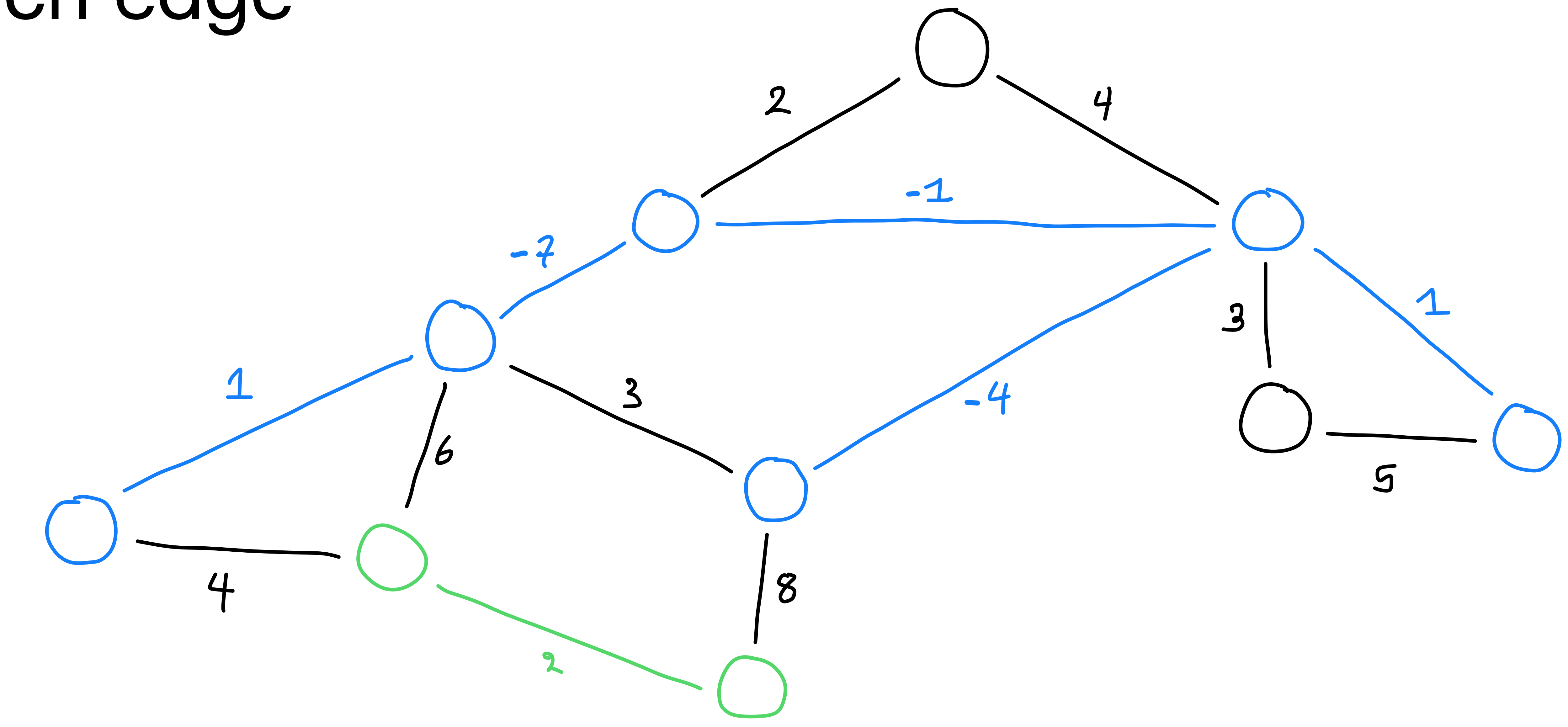
- The key will be finding a better graph.
- Consider a graph with vertex set $V_2 = V \times \{0,1\}$ with a vertex $(v, b) \in V_2$ indicating location x [whether store has been visited]
- **Starting vertex:** $(s,0)$ since item isn’t acquired
- **End vertex:** $(t,1)$ since item is acquired
- Now we are looking for shortest path from $(s,0)$ to $(t,1)$ in graph $G_2 = (V_2, E_2)$. But what is E_2?



Kruskal’s algorithm

High level

- Start with $F = (V, E' = \varnothing)$
- While there exists edges $e \in E \backslash E'$ such that $E' \cup \{e\}$ contains no cycles, add such edge of minimal weight $w(e)$ to E'



Analysis for divide and conquer runtimes

The master theorem

- For solving recursive equations of the form

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^k) \text{ and } T(< b) = O(1)$$

- Different cases based on how $f(n)$, a, and b compare:
    - If $a < b^k$, then $T(n) = O(n^k)$
    - If $a = b^k$, then $T(n) = O(n^k \log n)$
    - If $a > b^k$, then $T(n) = O(n^{\log_b a})$

Strassens's algorithm

7 multiplications + 10 additions

1 mult + 2 additions → M1

1 mult + 1 addition → M2

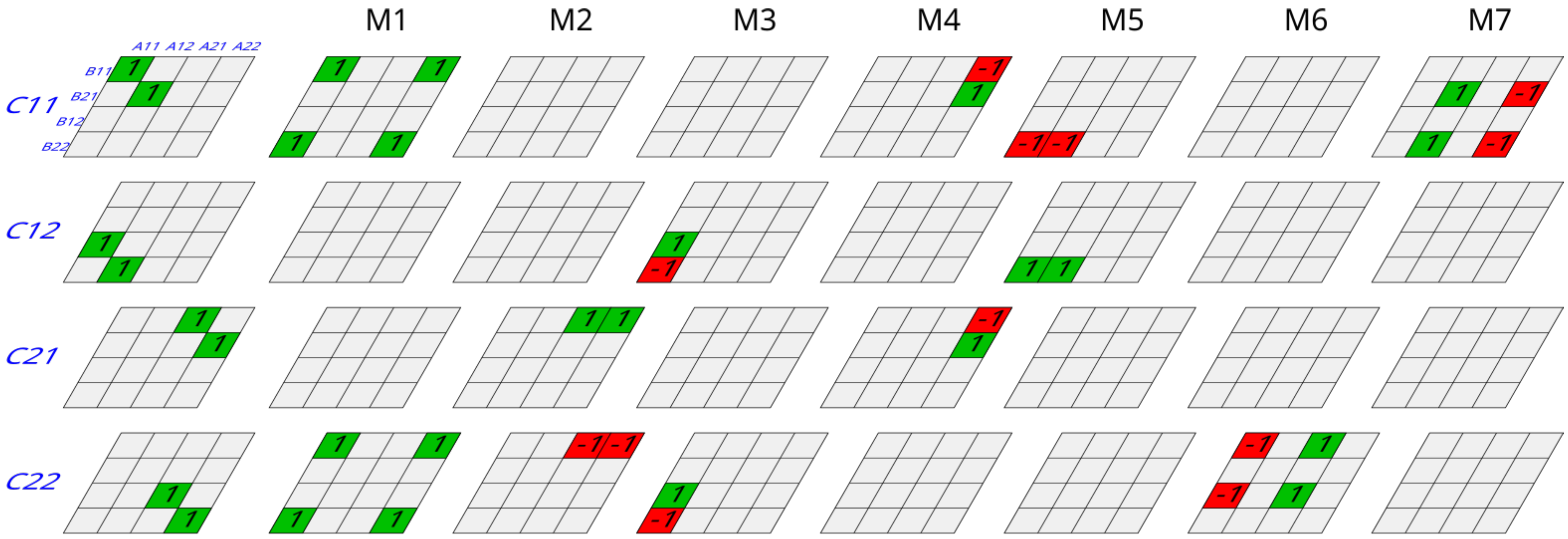
1 mult + 1 addition → M3

1 mult + 1 addition → M4

1 mult + 1 addition → M5

1 mult + 2 addition → M6

1 mult + 2 additions → M7



$$C_{11} = M_1 + M_4 - M_5 + M_7$$

$$C_{12} = M_3 + M_5$$

$$C_{21} = M_2 - M_4$$

$$C_{22} = M_1 - M_2 + M_3 + M_6$$

8 additions

Wikipedia article for Strassen's algorithm

$$M_1 = (A_{11} + A_{22})(B_{11} + B_{12})$$

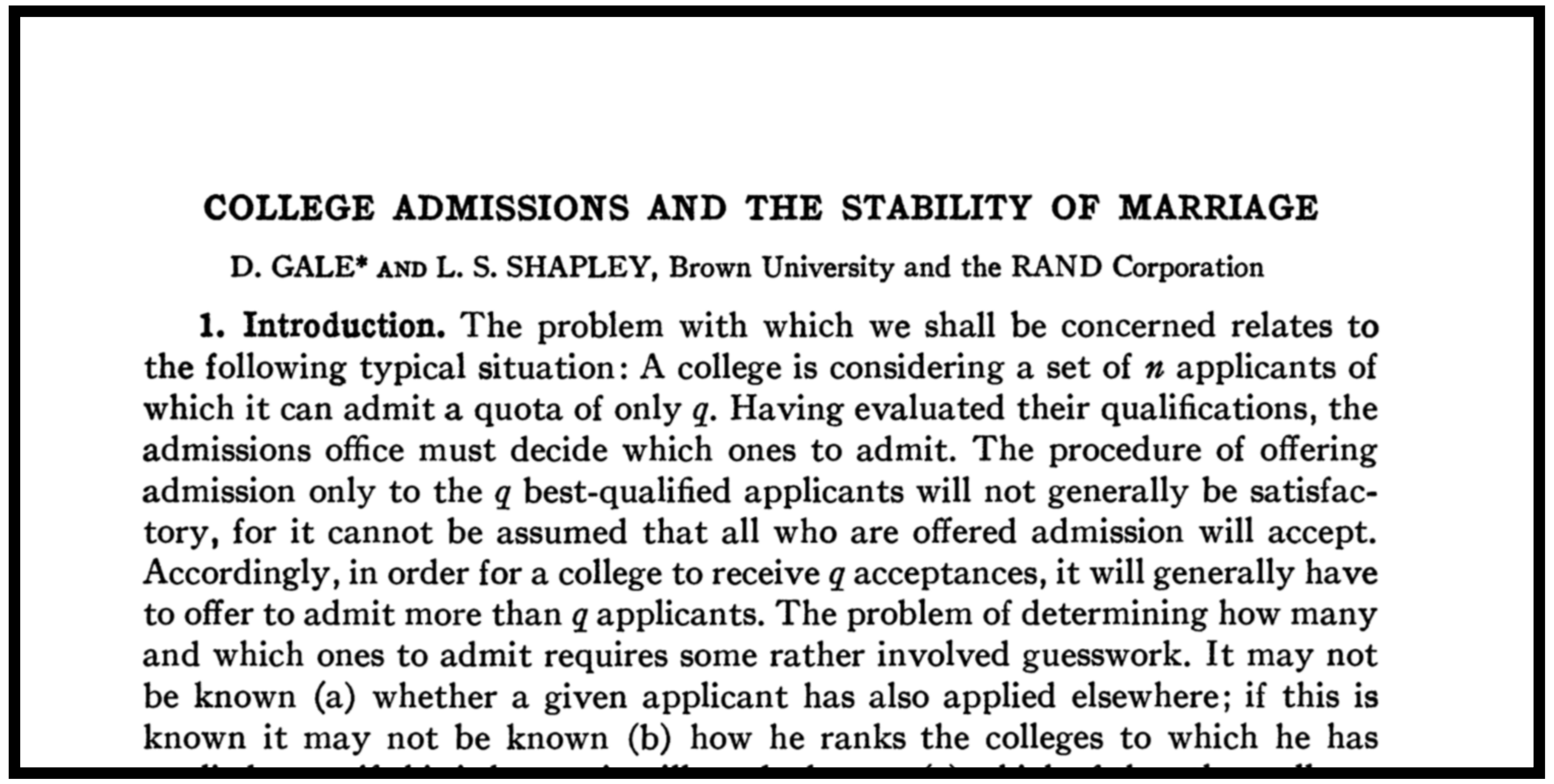
$$M_5 = (A_{11} + A_{12})B_{22}$$

$$M_4 = A_{22}(B_{21} - B_{11})$$

The history of the propose and reject algorithm

Gale and Shapley 1962

- The original paper was about n men and n women and a heterosexual notion of marriage.
- Gale and Shapley's algorithm defined the proposers as the men and the receivers as the women.
  - We will see next that the GS algorithm is **proposer**-optimal but not **receiver**-optimal.
  - For obvious reasons, we changed the notation.
  - As originally stated, the GS algorithm favored being a man. This social implication was not recognized for some time!
- Is fairness possible? In some cases, yes. But this is an active area of research!



COLLEGE ADMISSIONS AND THE STABILITY OF MARRIAGE

D. GALE* AND L. S. SHAPLEY, Brown University and the RAND Corporation

1. Introduction. The problem with which we shall be concerned relates to the following typical situation: A college is considering a set of n applicants of which it can admit a quota of only q. Having evaluated their qualifications, the admissions office must decide which ones to admit. The procedure of offering admission only to the q best-qualified applicants will not generally be satisfactory, for it cannot be assumed that all who are offered admission will accept. Accordingly, in order for a college to receive q acceptances, it will generally have to offer to admit more than q applicants. The problem of determining how many and which ones to admit requires some rather involved guesswork. It may not be known (a) whether a given applicant has also applied elsewhere; if this is known it may not be known (b) how he ranks the colleges to which he has

Shapley winning the 2012 Economics Nobel Prize (with Roth)

General dynamic programming algorithm

- **Iterate through subproblems:** Starting from the "smallest" and building up to the "biggest." For each one:
    - Find the optimal value, using the previously-computed optimal values to smaller subproblems.
    - Record the choices made to obtain this optimal value. (If many smaller subproblems were considered as candidates, record which one was chosen.)
- **Compute the solution:** We have the value of the optimal solution to this optimization problem but we don't have the actual solution itself. Use the recorded information to actually reconstruct the optimal solution.

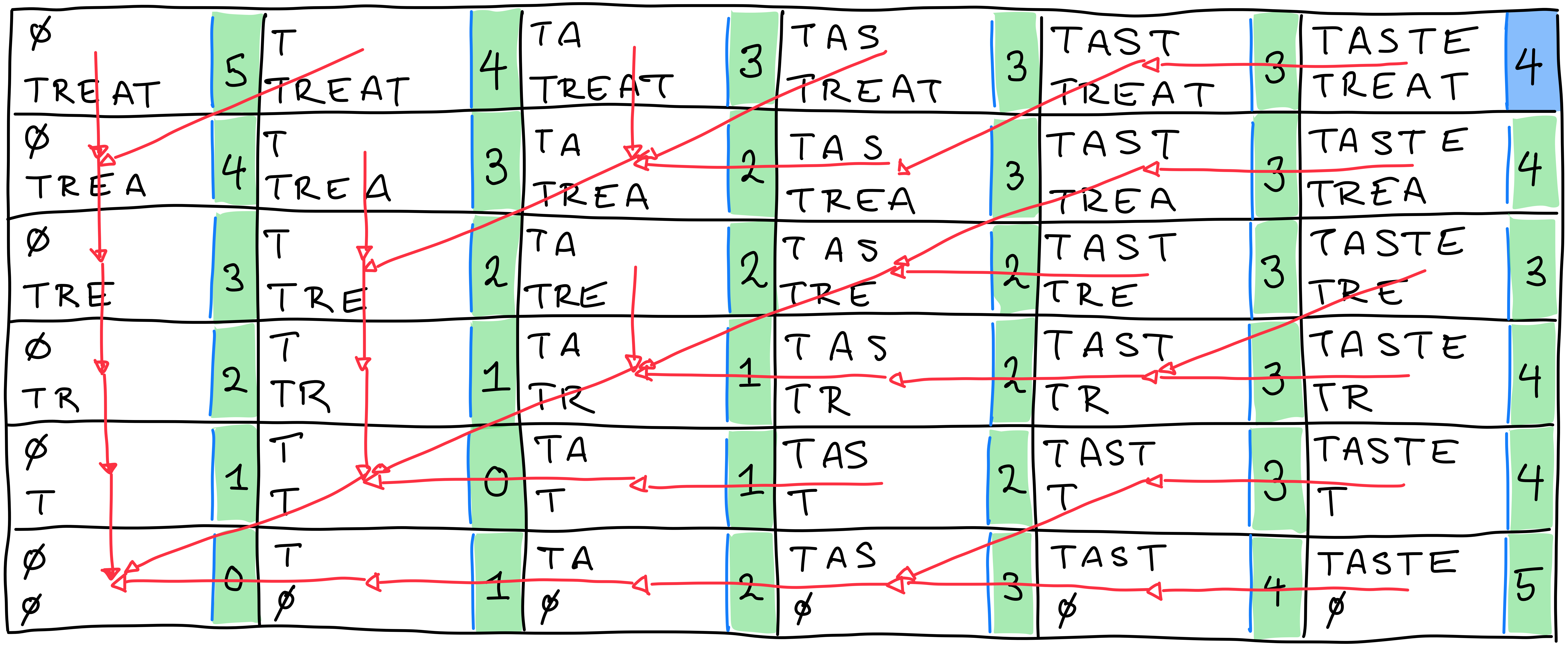
X = TASTE

L13

Edit distance walkthrough

TASTE v TREAT

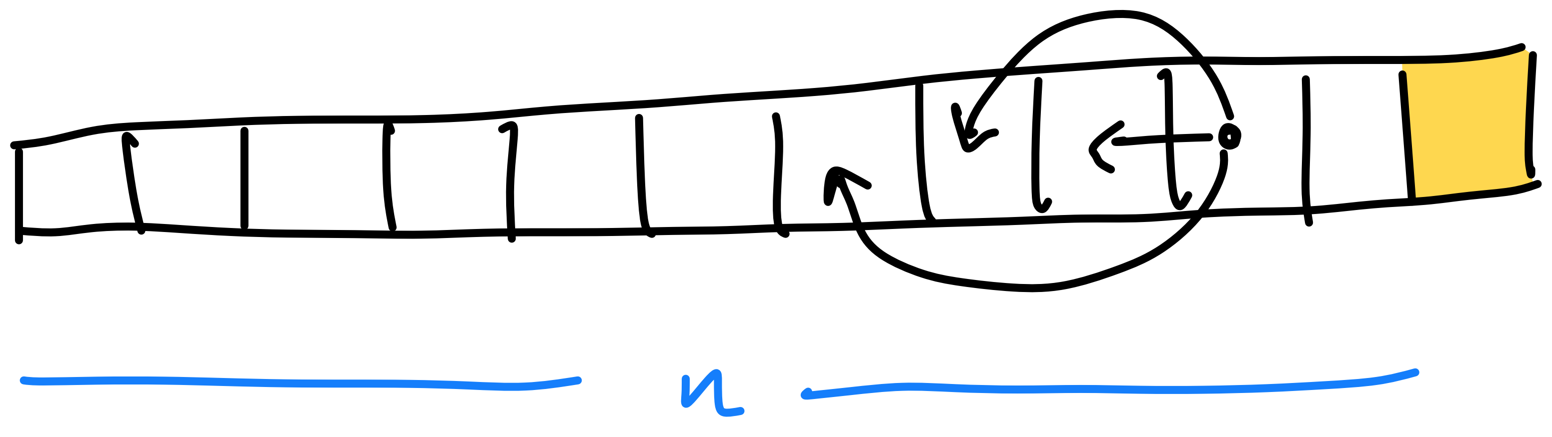
← = Delete, ↓ = Insert, ↙ = Equal or Substitute



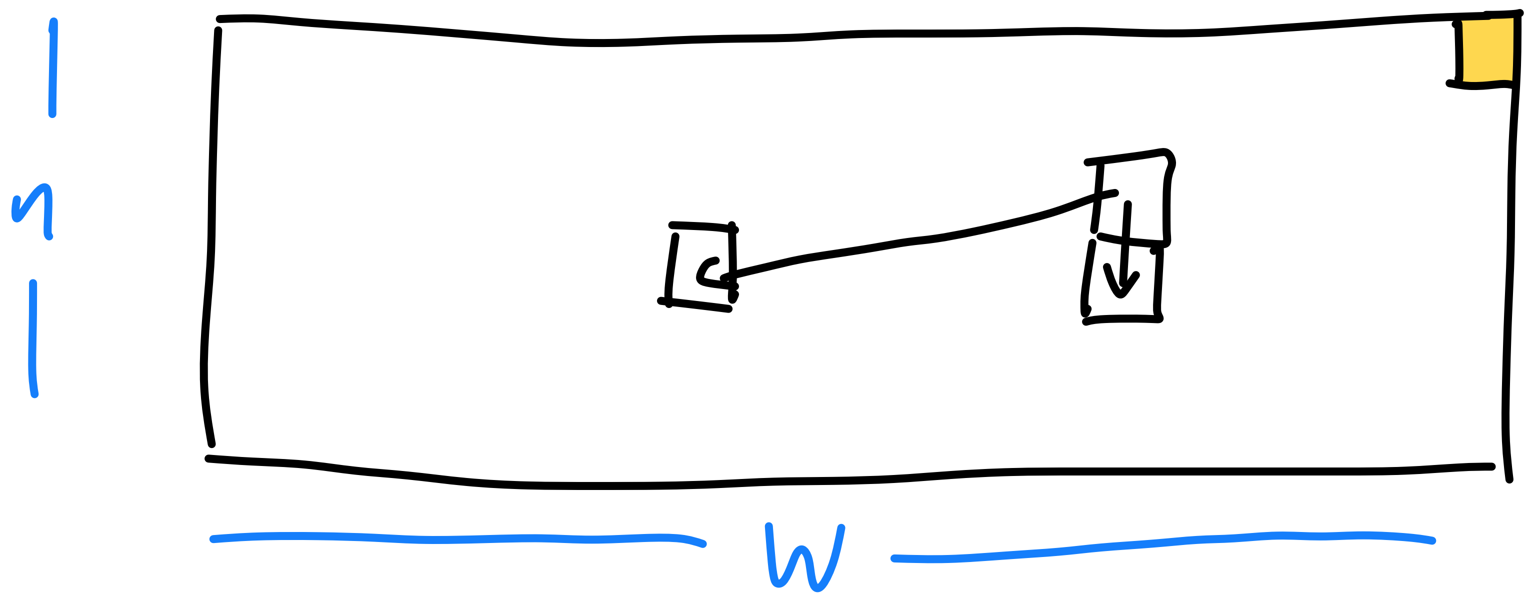
∅ TREAT	5	T TREAT	4	TA TREAT	3	TAS TREAT	3	TAST TREAT	3	TASTE TREAT	4
∅ TREA	4	T TREA	3	TA TREA	2	TAS TREA	3	TAST TREA	3	TASTE TREA	4
∅ TRE	3	T TRE	2	TA TRE	2	TAS TRE	2	TAST TRE	3	TASTE TRE	3
∅ TR	2	T TR	1	TA TR	1	TAS TR	2	TAST TR	3	TASTE TR	4
∅ T	1	T T	0	TA T	1	TAS T	2	TAST T	3	TASTE T	4
∅ ∅	0	T ∅	1	TA ∅	2	TAS ∅	3	TAST ∅	4	TASTE ∅	5

Dynamic programming patterns

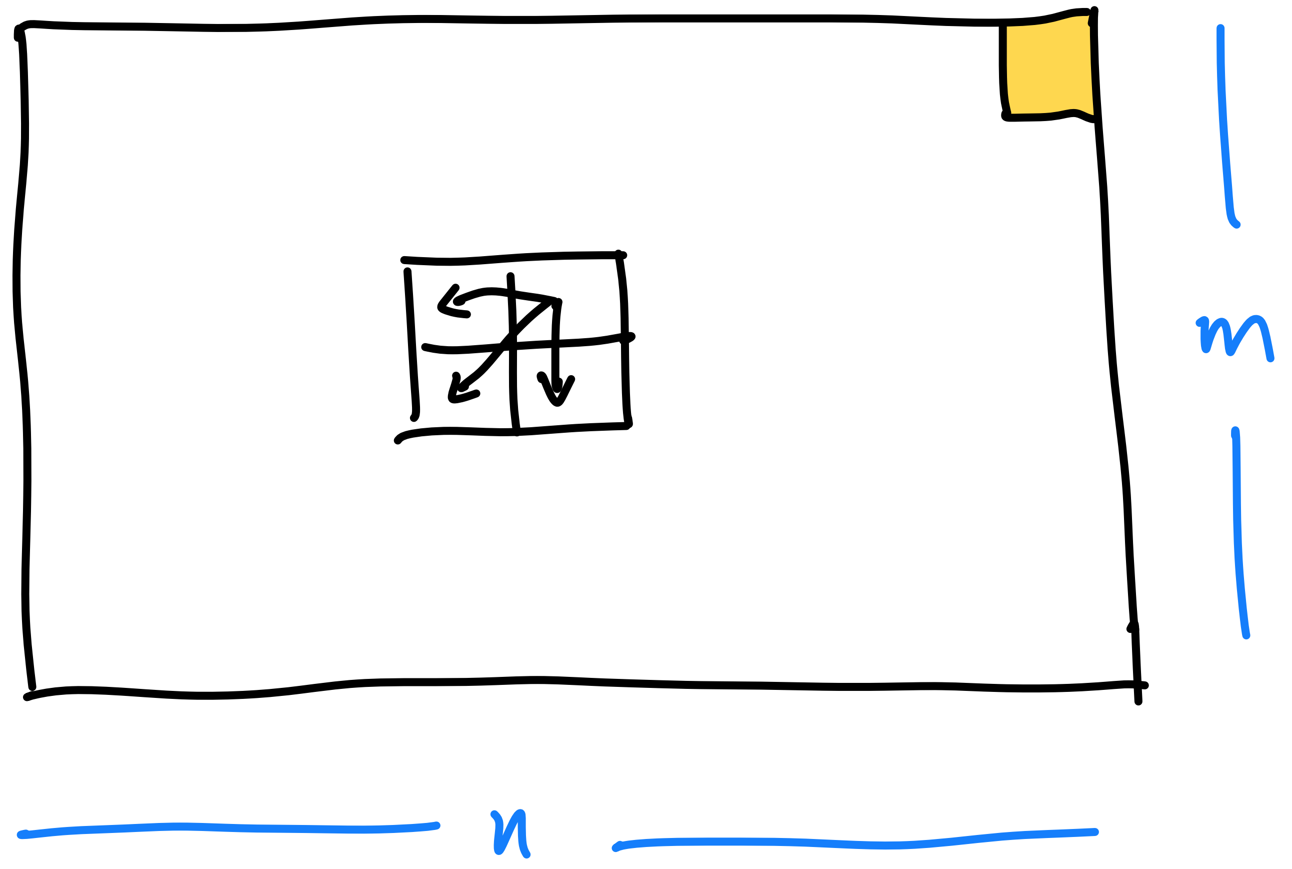
Tribonacci



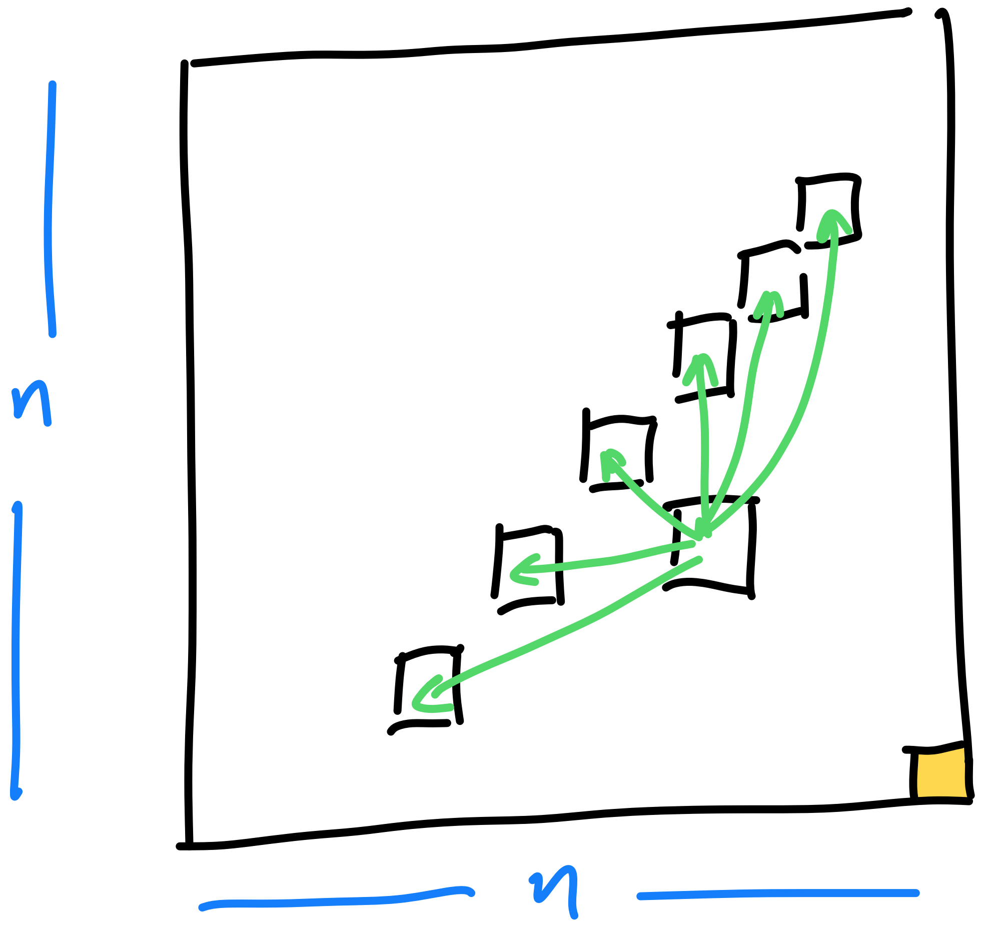
Knapsack



Edit distance

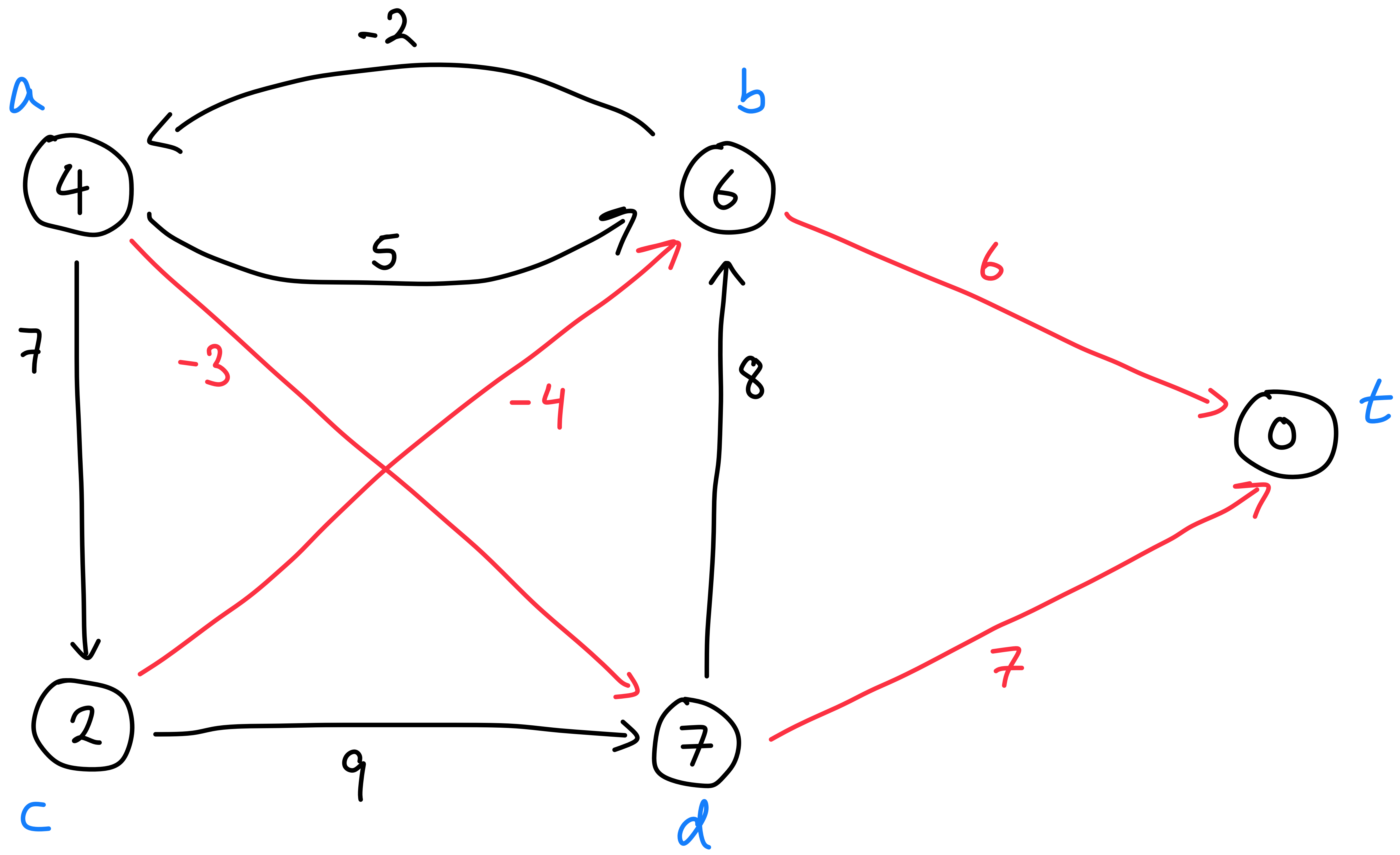


RNA second sequence



O(n) recursive calls per entry

Bellman-Ford example



Queue

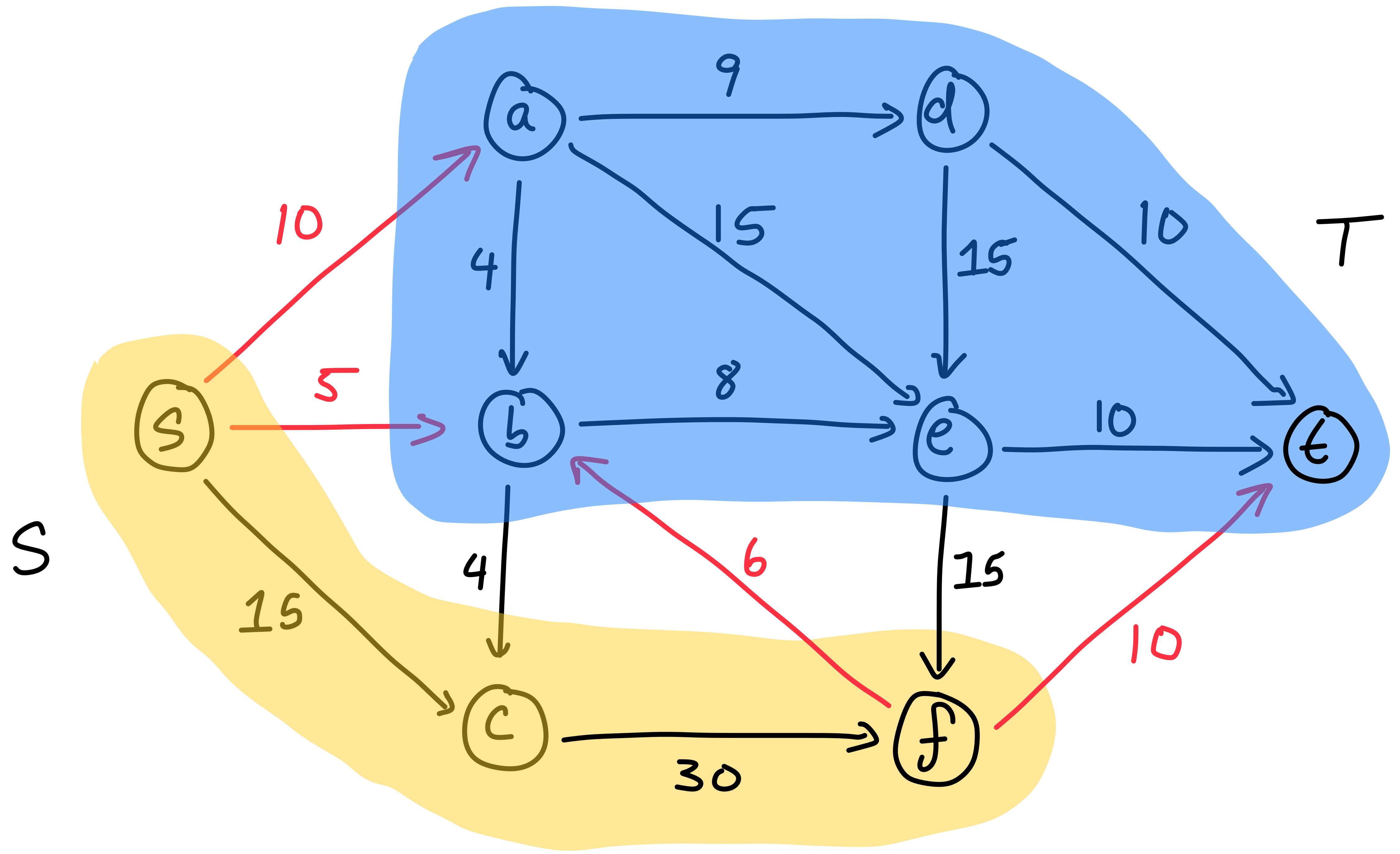
~~t~~
~~⊥~~
~~b~~
~~d~~
~~⊥~~
a
c
⊥

Graph cuts

- An **s-t cut** in a graph is a partition of the vertices into $V = S \sqcup T$ such that $s \in S$ and $t \in T$. The capacity of a s-t cut (S, T) is

$$c(S,T) := \sum_{\substack{\text{edges } e \text{ leaving} \\ S}} c(e)$$

$$c(S,T) = 10 + 5 + 6 + 10 = 21$$

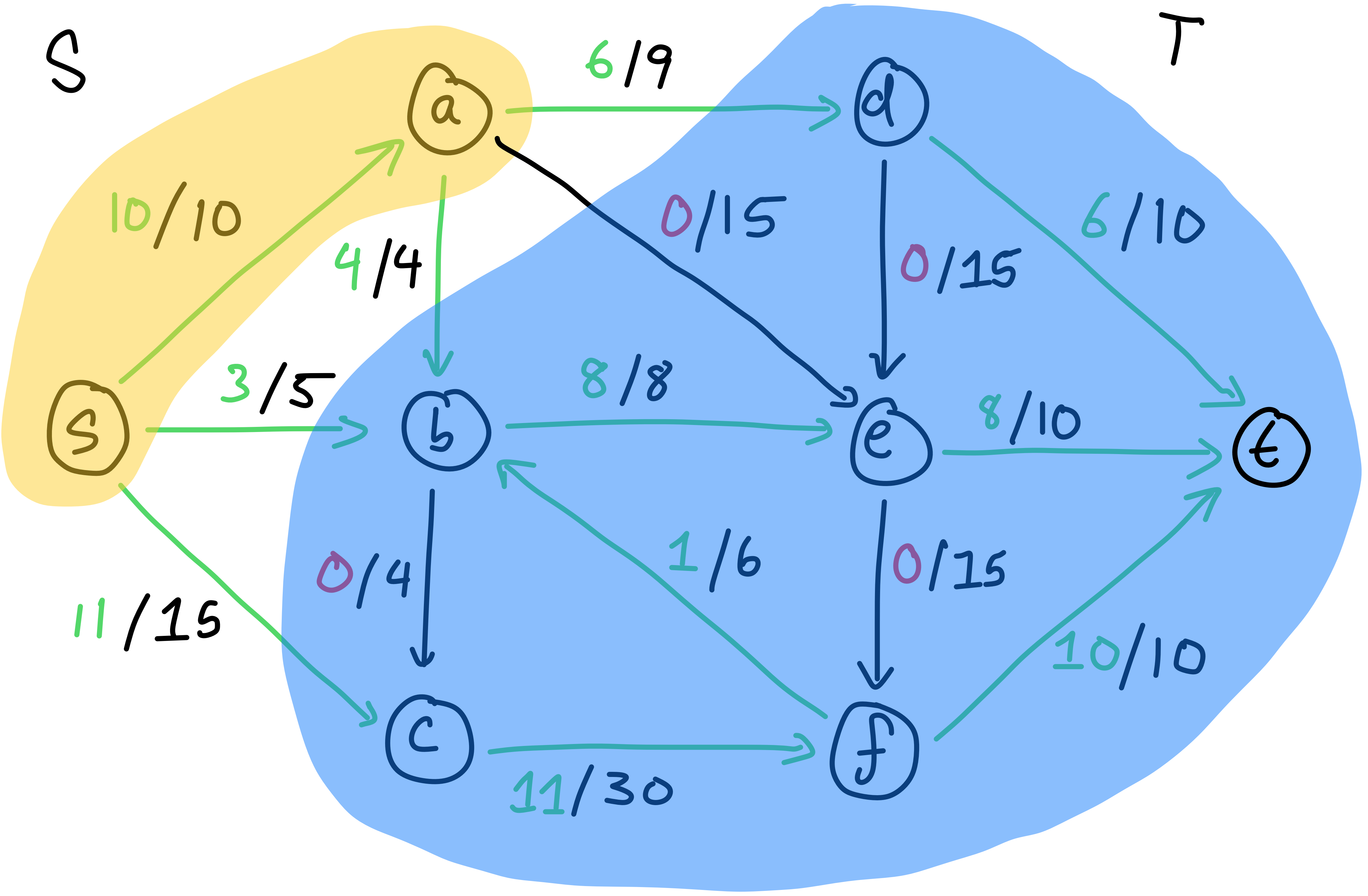


Flow value lemma

- **Flow value lemma:** Let f be a s-t flow and *any* s-t cut (S, T). Then

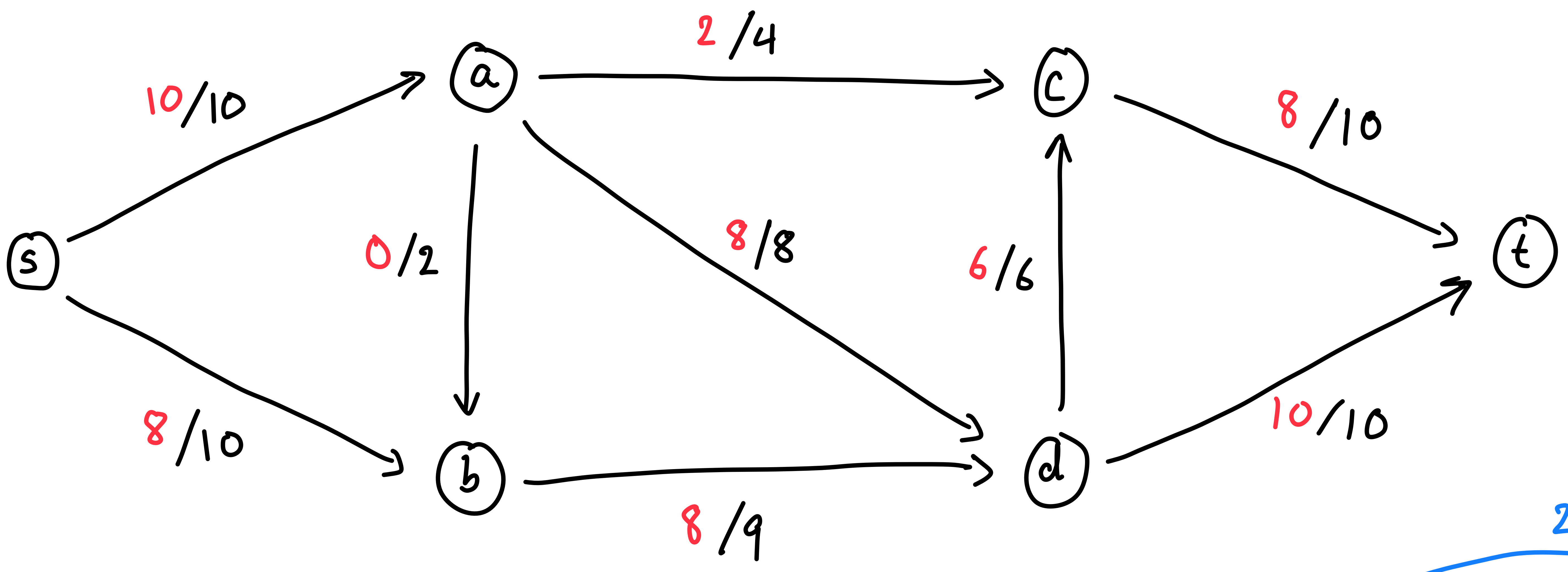
$$v(f) = \sum_{e \text{ from } S \text{ to } T} f(e) - \sum_{e \text{ from } T \text{ to } S} f(e)$$

- **Proof (intuition):**
  - Add the vertices of S one by one until the set is generated.

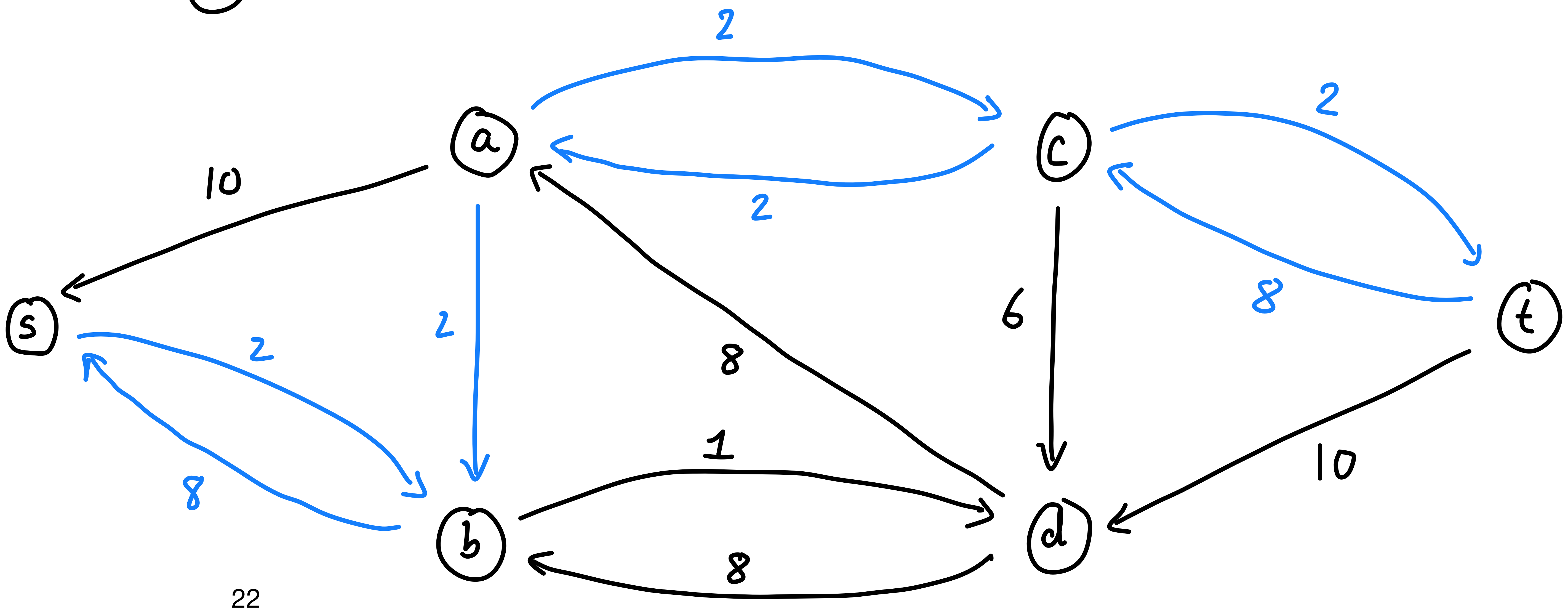


Ford-Fulkerson animation

Graph G and flow f:



Residual graph G_f:



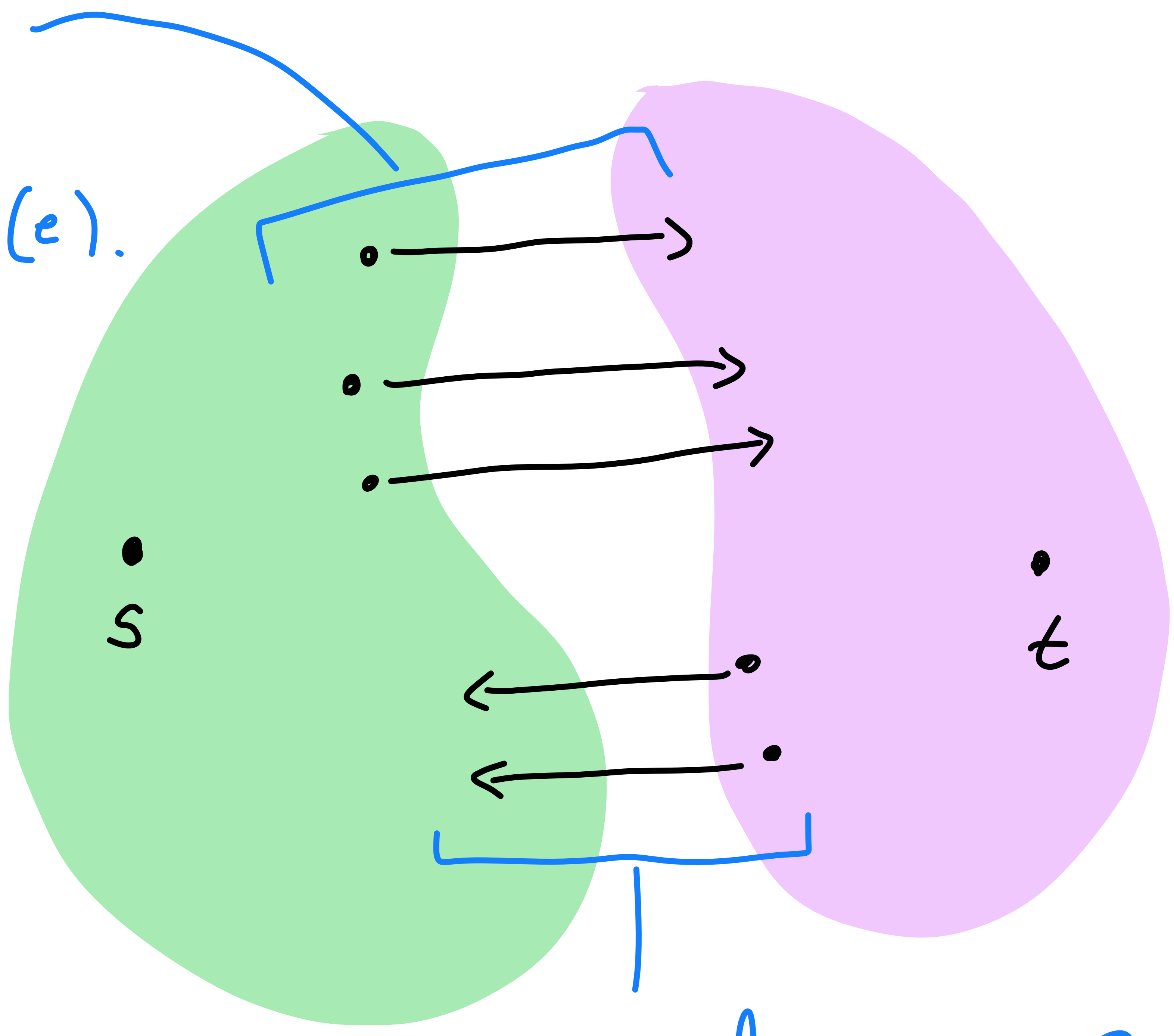
The max flow/min cut theorem

(3) $\Longrightarrow$ (1)

- **Proof:**
  - Edges from S to T are *saturated* with flow.
  - Edges from T to S have no flow.
  - $v(f) = f^{\text{out}}(S) - f^{\text{in}}(S)$
  - $$= \sum_{e \text{ from } S \text{ to } T} f(e) - \sum_{e' \text{ from } T \text{ to } S} f(e')$$
  - $$= \underbrace{\sum_{e \text{ from } S \text{ to } T} c(e)}_{C(S,T)} - \underbrace{\sum_{e' \text{ from } T \text{ to } S} 0}_{0}$$
  - $= C(S, T).$

original graph G

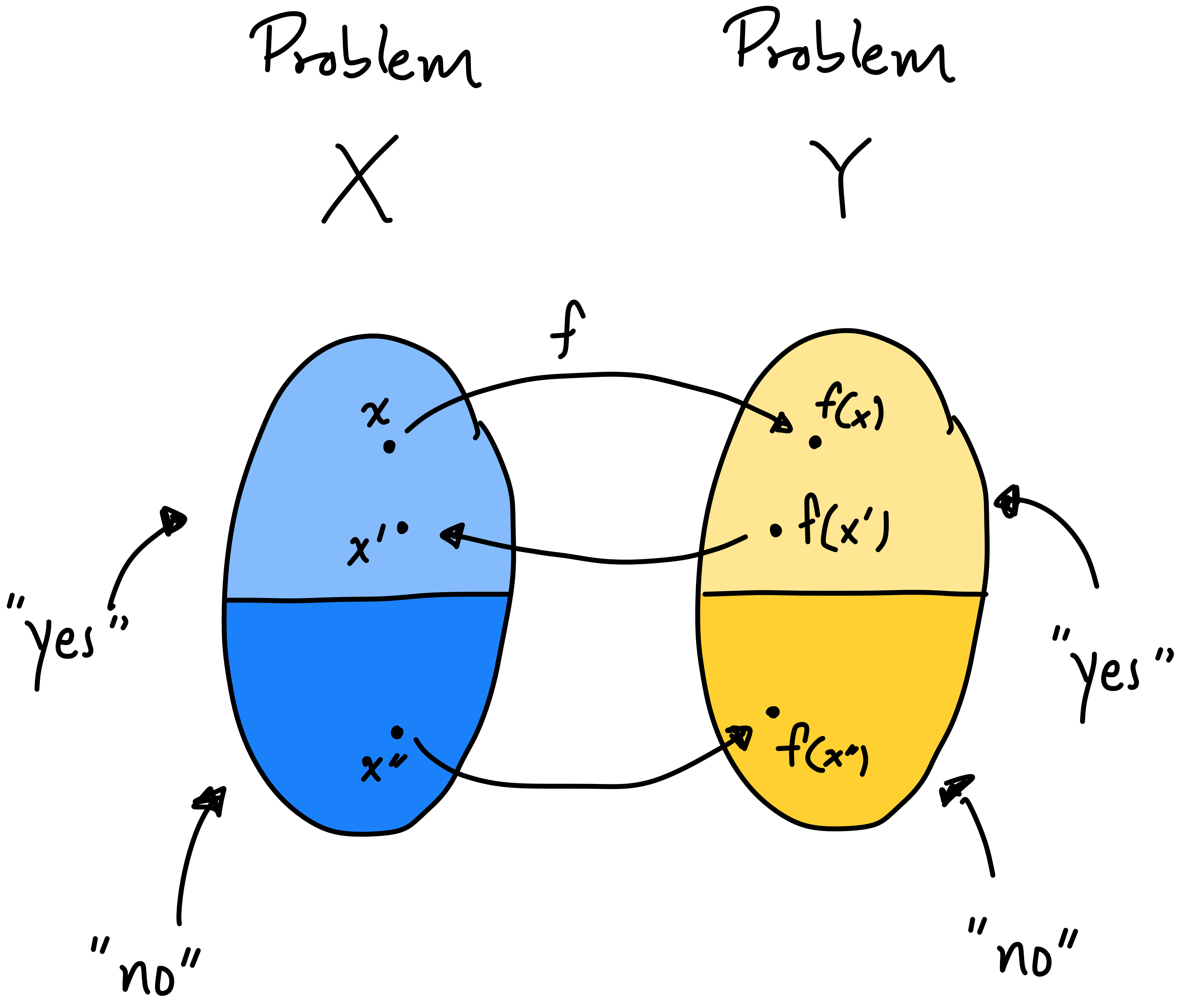
all edges from S to T satisfy $f(e) = c(e)$.



all edges from T to S satisfy $f(e') = 0$.

Proving a reduction is correct

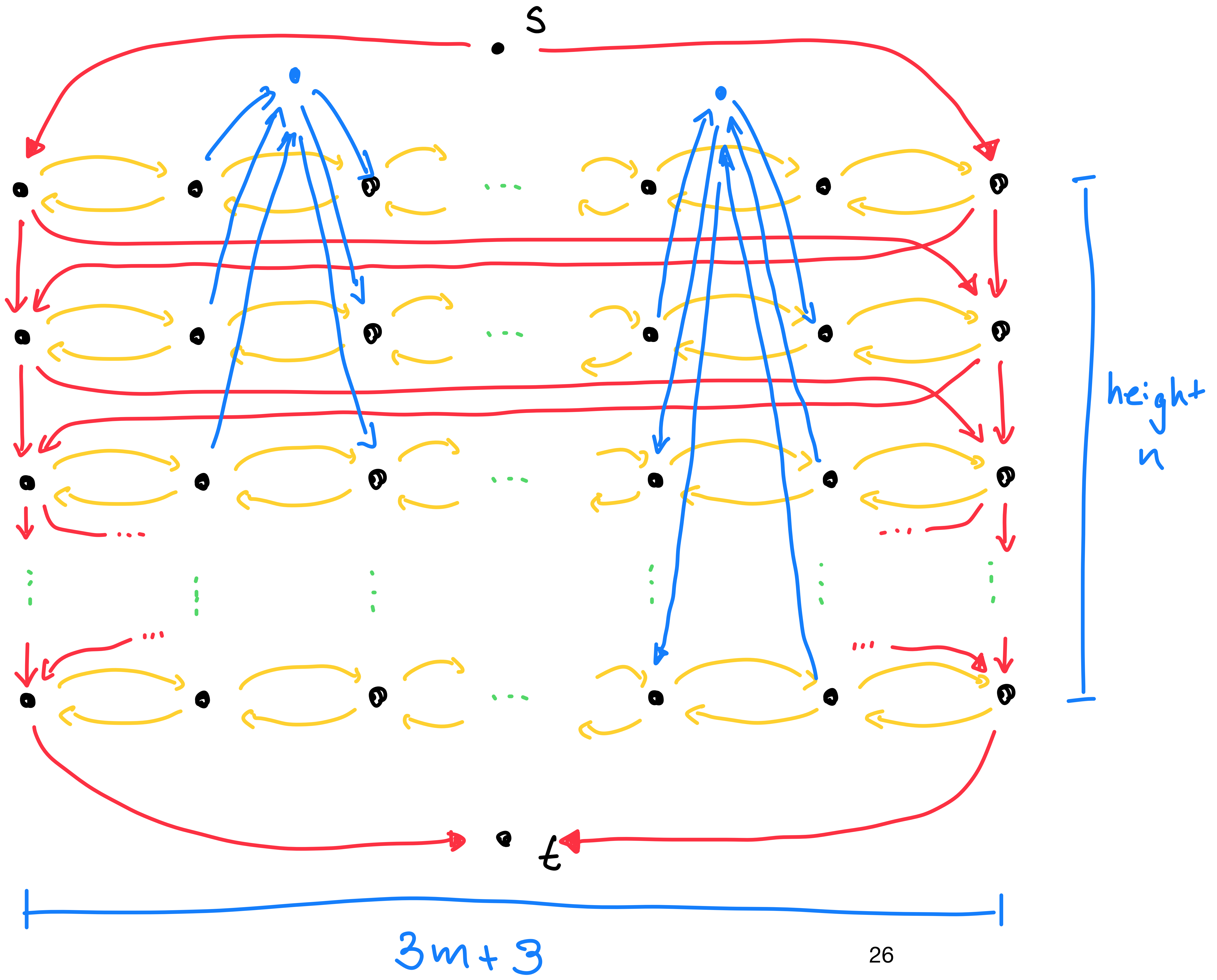
- The previous example is a reduction between Subset Sum and D-Knapsack
- To generate a reduction $X \leq_p Y$ between two decision problems X and Y
  - We need to find a **poly-time computable** function $f : X \to Y$ that converts instances x of X into instances $f(x)$ of Y
    - Morally, f is the precompute we would apply before the algorithm for Y
  - If for every $x \in X$ that is a "yes", then $f(x) \in Y$ is also a "yes" instance
  - If for every $f(x') \in Y$ that is a "yes", then $x' \in X$ is also a "yes" instance
    - Equiv. to: If $x'' \in X$ is a "no", then $f(x'') \in Y$ is a "no"



Proving more NP-complete problems

- **Recipe** for showing that problem Y is NP-complete
  - Step 1: Show that $Y \in$ NP.
  - Step 2: Choose a known NP-complete problem X.
  - Step 3: Prove that $X \leq_p Y$.
- **Correctness** of recipe: We claim that $\leq_p$ is a transitive operation.
  - If $W \leq_p X$ and $X \leq_p Y$ then $W \leq_p Y$.
  - For any problem $W \in$ NP, then $W \leq_p Y$, proving that Y is NP-complete.

Hamiltonian cycle is NP-complete



$m = \#$ of clauses

Install gadget for clause j

$\varphi_j = (x_{i_1} \vee x_{i_2} \vee x_{i_3})$

in columns $3j, 3j-1$

and rows i_1, i_2, i_3

so that gadgets never overlap

Linear programming

- An optimization problem paradigm
- Both the optimization function f and feasible region Γ are linear.

$$\max \underset{\underset{\mathbb{R}^n}{\uparrow}}{\begin{bmatrix} — c — \end{bmatrix}} \cdot \underset{\underset{\mathbb{R}^n}{\uparrow}}{\begin{bmatrix} | \\ x \\ | \end{bmatrix}} \quad \text{subject to} \quad \underset{\underset{\mathbb{R}^{m \times n}}{\uparrow}}{\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix}} \cdot \begin{bmatrix} | \\ x \\ | \end{bmatrix} \le \underset{\underset{\mathbb{R}^m}{\uparrow}}{\begin{bmatrix} | \\ b \\ | \end{bmatrix}} \quad \text{and} \quad \begin{bmatrix} | \\ x \\ | \end{bmatrix} \ge 0.$$

Global space $\Sigma = \mathbb{R}^n$, feasible region $\Gamma = \{x : Ax \le b, x \ge 0\}$, opt. fn. $f(x) = c^T x$

Linear programming duality

$$\max \quad S_1 x_1 + S_2 x_2$$

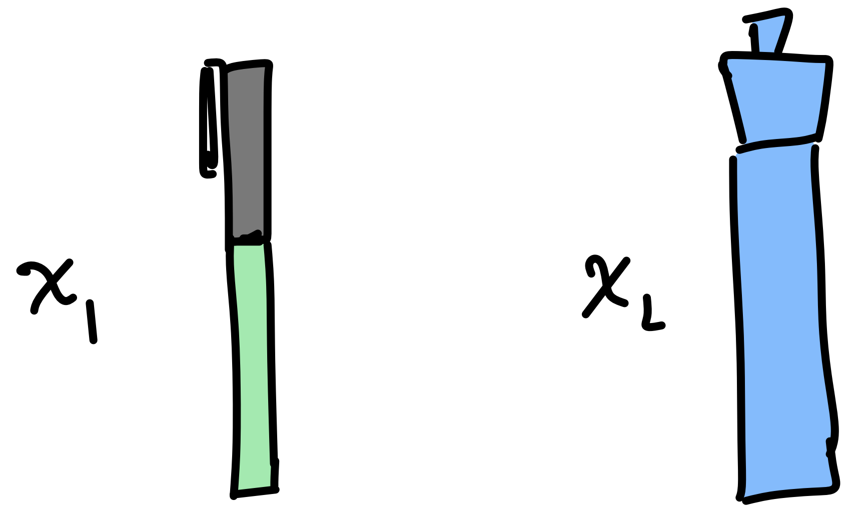
$$\text{s.t.} \quad L_1 x_1 + L_2 x_2 \leq L$$

$$I_1 x_1 + I_2 x_2 \leq I$$

$$P_1 x_1 + P_2 x_2 \leq P$$

$$x_1, x_2 \geq 0.$$

number of pens & markers



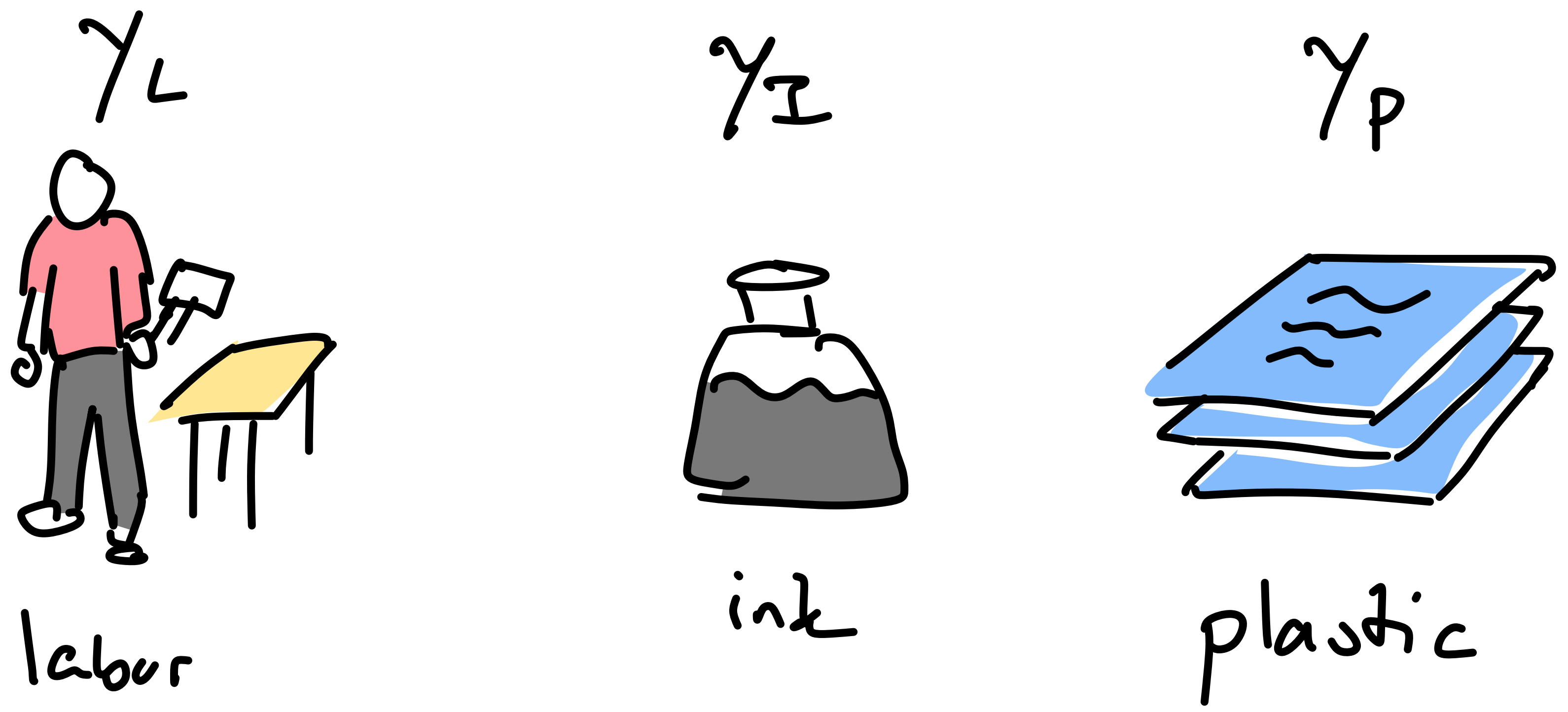
$$\min \quad y_L L + y_I I + y_P P$$

$$\text{s.t.} \quad y_L L_1 + y_I I_1 + y_P P_1 \geq S_1$$

$$y_L L_2 + y_I I_2 + y_P P_2 \geq S_2$$

$$y_L, y_I, y_P \geq 0.$$

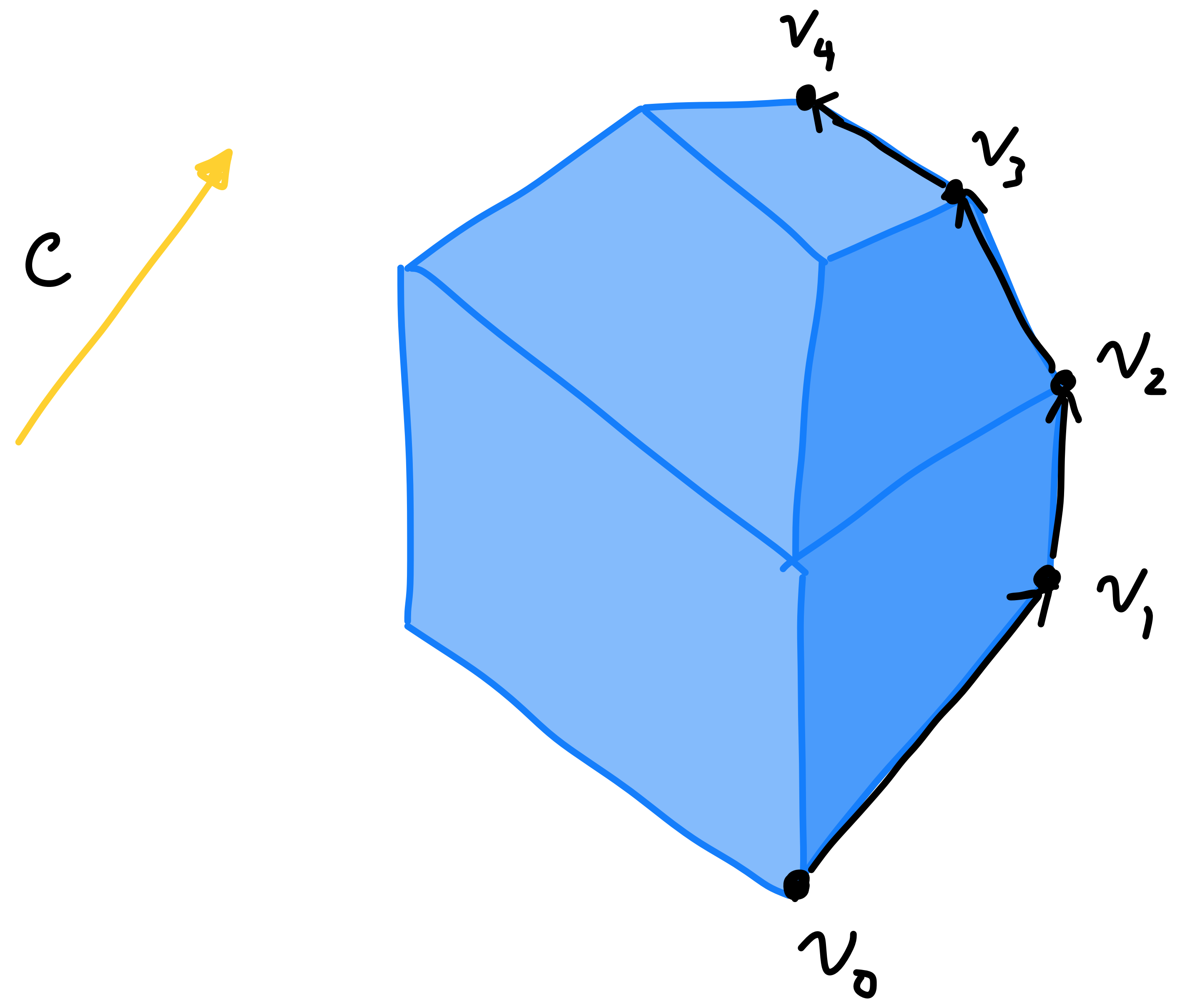
price of materials



The simplex method

- Simplex is a greedy algorithm
- **High-level algorithm**:
  - Start from a vertex of the polytope
  - In each step, move to the neighboring vertex that optimizes $c^\top x$
  - Equivalently, move along the edge pointing the most in the c direction

all edges point away from c, so alg halts



That brings us to today!

Setup of the final

- 4 long-form questions each worth ~10 points
- The problems are increasingly harder (but take a look at all of them, first)
- Cheat sheet is allowed: one page, double sided
- Topics not covered: stable matching and linear programming
- Location: Here, G20
- Time: Monday March 16th 2:30 - 4:20

Tips for the final

- Slow down! Nothing requires an extensive amount of writing.
- Use scratch paper to sketch your whole argument/algorithm/proof
    - Try out on a small example or two before writing in booklet
    - Only write what you need in the booklet
- Read the question carefully and answer only what you need
- It is unlikely that there are mistakes, but if you have confusions, ask us.

What’s next?

What should you do next?

- **Interview prep**: probably a good idea to practice **coding** algorithms you have seen in this course, and new algorithms that you design to solve other problems.
- This course is an introduction to many concepts. To develop more familiarity with them (and have them become second nature), coding is one good way. This will help with also $O(1)$ optimizations that are very important in practice!
- All your hard work on proofs of correctness is not in vain. Being able to concretely explain why your algorithm is correct is an important skill both in terms of interviews as well as algorithmic practice.

What else is there in computer science theory?

- **CSE422 Advanced Toolkit for Modern Algorithms**
    - Prerequisite: CSE 312; CSE 332; and MATH 208.
    - Learn more modern algorithms particularly for data analysis, streaming/online algorithms, and statistical learning. Most algorithms are randomized and very common in practice.

What else is there in computer science theory?

- **CSE431 Introduction to Theory of Computation**
  - Prerequisite: CSE 312.
  - Learn more about which provables have algorithmic *lower bounds* and compare different models of computation such as time, space, communication, non-determinism, etc.

What else is there in computer science theory?

- **CSE426 Cryptography**
  - Prerequisite: CSE 312.
  - Learn about how computationally difficult problems can be transformed into the backbone of our modern security systems. Reductions will become second nature after this course!

What else is there in computer science theory?

- **CSE434 Introduction to quantum computation!!!**
  - Prerequisite: CSE 312 and MATH 136 or 208.
  - Learn about how to compute with qubits, the basis of quantum algorithms, and what problems quantum computers could and couldn't solve. Some programming assignments but mostly a theory course.

What else is there in computer science theory?

- **CSE521, 522, or 525 Graduate algorithms**
  - Prerequisite: by request.
  - Learn about how randomization, stochastic processes, and approximation theory interact with all the classic algorithms we learned in this course. Approach each problem from a new mathematical perspective.

What else is there in computer science theory?

- **Other graduate courses**
  - CSE526 Cryptography
  - CSE531 Complexity theory
  - CSE534 Quantum information and computation (I teach this!)
  - CSE535 Convex optimization
  - CSE599 Many topics courses (next quarter: communication complexity and quantum complexity theory)

My hopes for you

- You learned a lot and now approach life from a *computational* lens
- You are better at communicating your ideas and making arguments
- The high level principles of algorithm design stay with you

The end (congrats!)