Midterm 2

Name: Targaryen

SID: 0123456789

Name and SID of student to your left: Lannister

Name and SID of student to your right: Stark

Exam Room:

Rules and Guidelines

- **The exam will last 110 minutes.**
- The exam has 115 points in total.
- Answer all questions. Read them carefully first. Not all parts of a problem are weighted equally.
- Write your student ID number in the indicated area on each page.
- Be precise and concise. **Write in the solution box provided.** You may use the blank page on the back for scratch work, but it will not be graded. Box numerical final answers.
- The problems may **not** necessarily follow the order of increasing difficulty. ***Avoid getting stuck on a problem.***
- Any algorithm covered in lecture can be used as a blackbox. Algorithms from homework need to be accompanied by a proof or justification as specified in the problem.
- Throughout this exam (both in the questions and in your answers), we will use ω_n to denote the first n^{th} root of unity, i.e., $\omega_n = e^{2\pi i/n}$.
- You may assume that comparison of integers or real numbers, and addition, subtraction, multiplication and division of integers or real or complex numbers, require $O(1)$ time.
- Good luck!

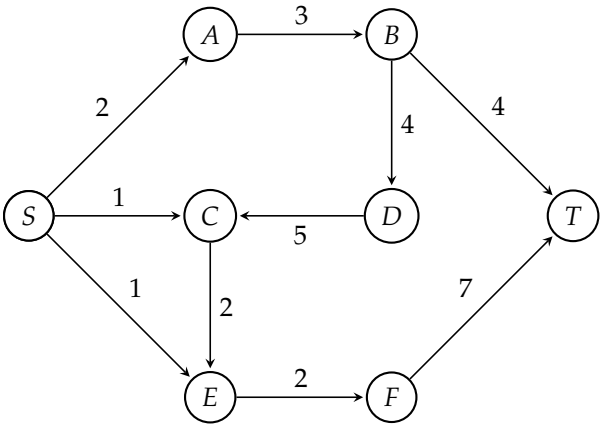
Discussion Section

Which section(s) do you attend? Feel free to choose multiple, or to select the last option if you do not attend a section. **Please color the checkbox completely. Do not just tick or cross the boxes.**

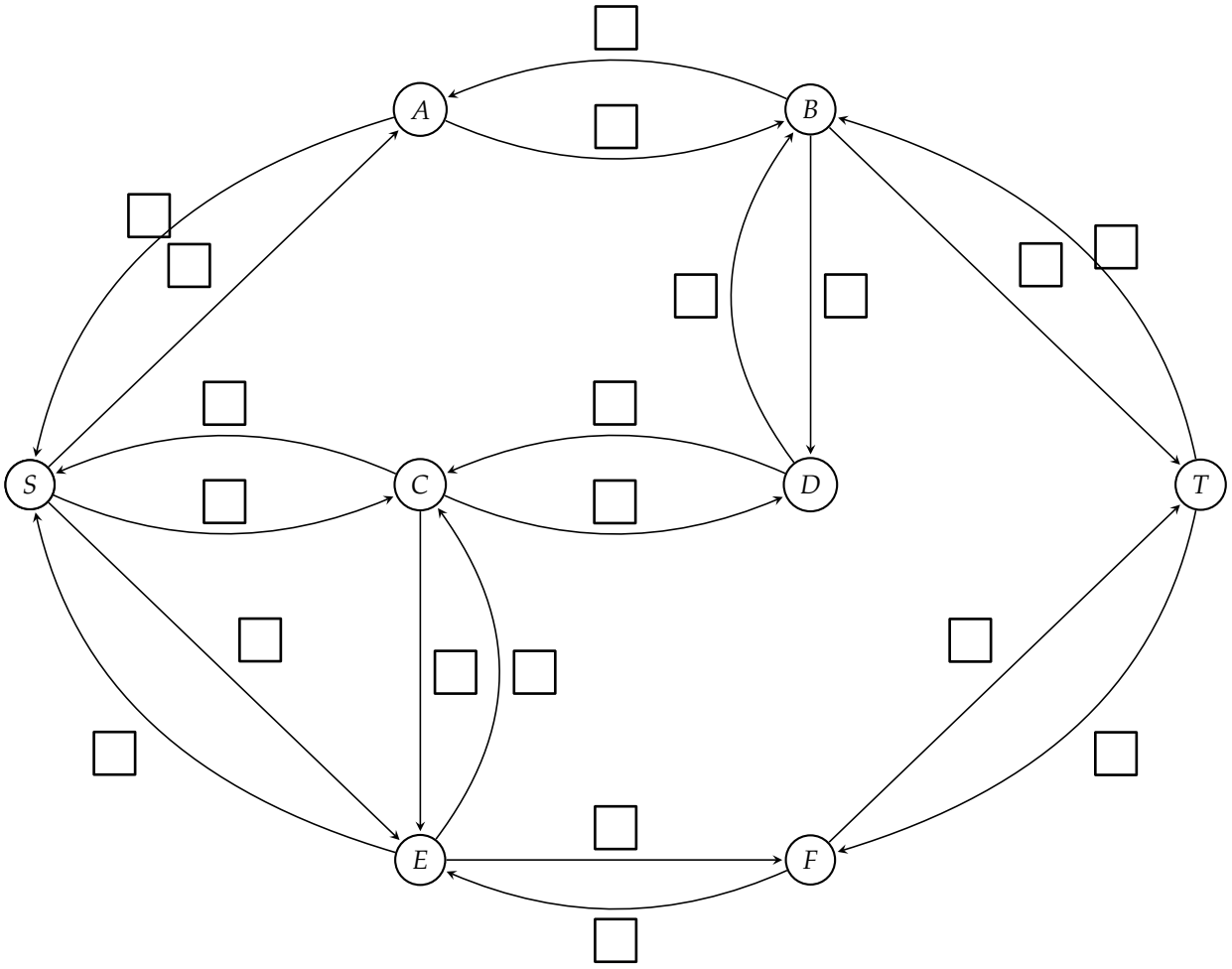
- ☐ Kevin Zhu, Wednesday 12 - 1 pm, Wheeler 200
- ☐ Andrew Che, Wednesday 1-2 pm, Dwinelle 79
- ☐ Kevin Li and Param (Exam Prep), Wednesday 1-2 pm, Wheeler 224
- ☐ Adnaan Sachidanandan, Wednesday 2-3 pm, Wheeler 108
- ☐ Wilson Wu, Wednesday 2-3 pm, Hearst Memorial Gym 242
- ☐ Cindy Zhang, Wednesday 3-4 pm, Cory 289
- ☐ Tyler Hou (Leetcode), Wednesday 4-5 pm, Etcheverry 3109
- ☐ Elicia Ye, Thursday 11-12 pm, Wheeler 130
- ☐ Cindy Zhang, Thursday 12-1pm, Remote
- ☐ Reena Yuan, Thursday 1-2pm, Etcheverry 3113
- ☐ Tynan Sigg, Thursday 4-5 pm, Soda 310
- ☐ Adnaan Sachidanandan (LOST), Thursday 5-7, Cory 258
- ☐ Video walkthroughs
- ☐ Don't attend Section

1 Max Flow (10 points)

Consider the execution of Ford-Fulkerson algorithm for Maximum Flow on the following network:



(i) In the first iteration of the Ford-Fulkerson algorithm, suppose it chooses the path $S \to A \to B \to D \to C \to E \to F \to T$. Write down the residual capacities of the edges after this iteration.



(ii) In the final flow output by the algorithm, what is the flow along these edges?

Edge	Flow Value
$A \to B$	
$D \to C$	
$E \to F$	
$C \to E$	

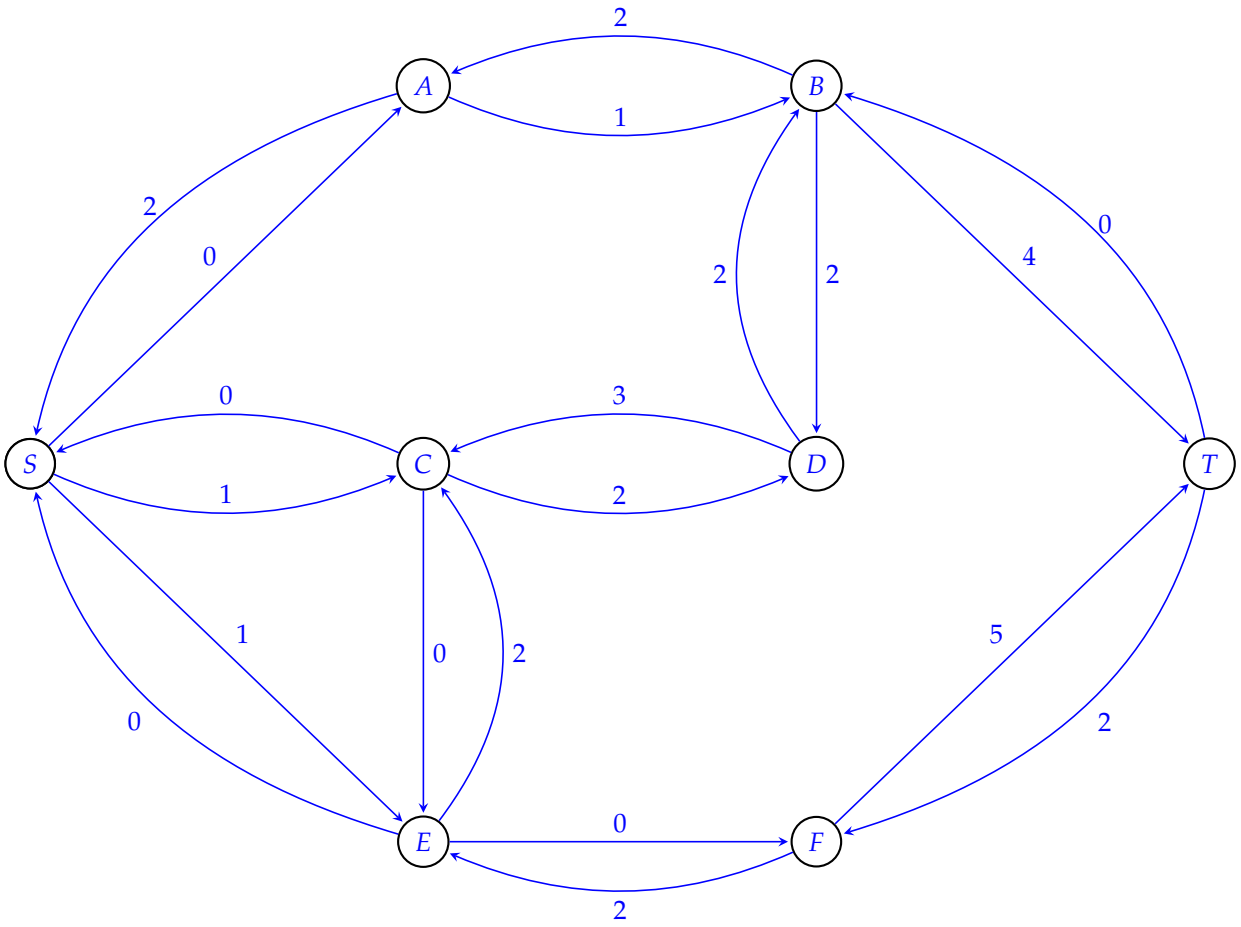
(iii) What is the value of the maximum flow on this network? | |

(iv) List all $s - t$ minimum cuts in the graph

Vertices on s side of the cut	Vertices on t side of the cut

Solution:

(i)



(ii)

Edge	Flow Value
$A \to B$	2
$D \to C$	0
$E \to F$	2
$C \to E$	1

(iii) Max flow has value 4.

(iv)

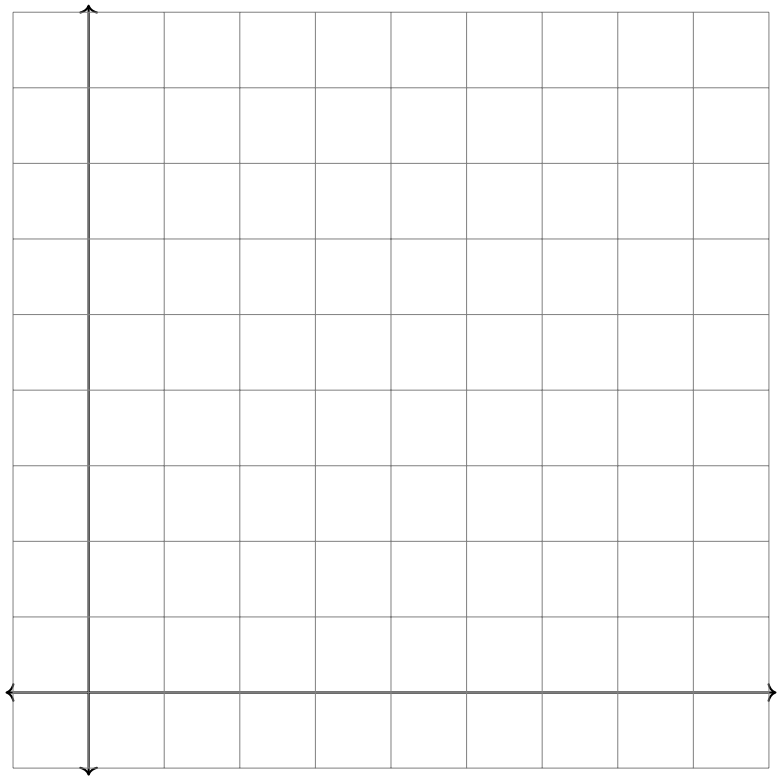
Vertices on s side of the cut	Vertices on t side of the cut
$\{S\}$	$\{A, B, C, D, E, F, T\}$
$\{S, C, E\}$	$\{A, B, D, F, T\}$

2 Linear Programming (11 points)

Consider the following linear program.

$$\begin{aligned} \text{Maximize} \quad & x \\ \text{Subject to} \quad & -x - y \leq -3 \\ & x - y \leq 3 \\ & x + y \leq 6 \\ & x, y \geq 0 \end{aligned}$$

(i) (**6 points**) Draw out the feasible region for the LP.

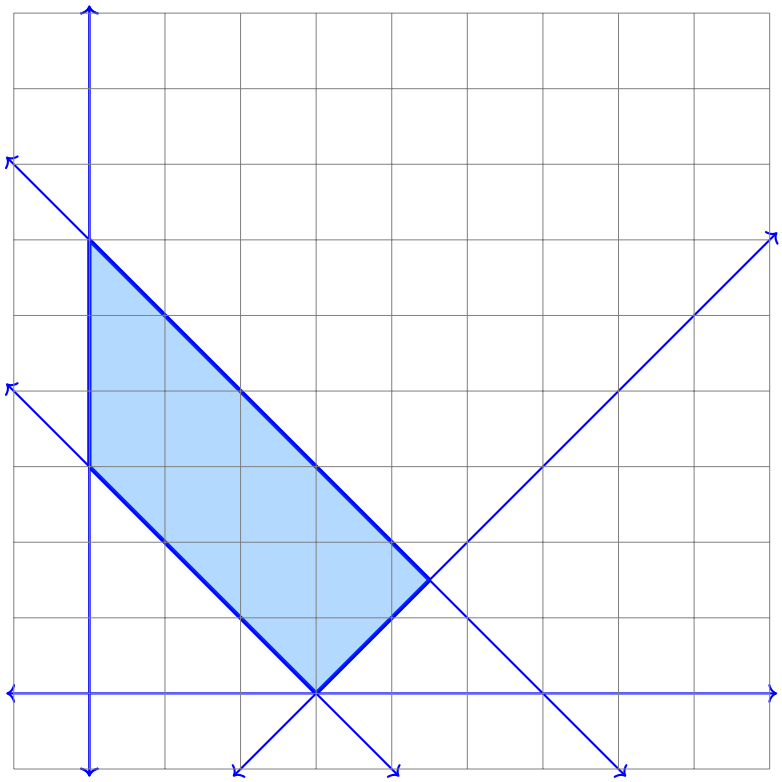


(ii) (**3 points**) Write the dual linear program (use variables a, b, c).

(iii) (**2 points**) Guess the optimal solution to the dual linear program. (Hint: recall that feasible solutions to the dual program correspond to the "proofs" of bounds on primal)

Solution:

(i)



(ii)

$$\begin{aligned} &\text{Minimize} && -3a + 3b + 6c \\ &\text{Subject to} && -a + b + c \geq 1 \\ & && -a - b + c \geq 0 \\ & && a, b, c \geq 0 \end{aligned}$$

(iii) a = 0, b = 1, c = 1.

3 Zero Sum (3 points)

Consider the zero-sum game given by the following 3×3 matrix (one of the payoffs is denoted by X).

X	10	20
40	30	10
20	40	10

Suppose the row player goes first and plays a mixed strategy given by,

$$\Pr[\text{ row } 1] = 0.5$$
$$\Pr[\text{ row } 2] = 0.2$$
$$\Pr[\text{ row } 3] = 0.3$$

If the column player's optimal response is not unique, what is the value of X?

(You can show your work in the box below for partial credit)

Solution: Expected payoffs are $0.5X + 14, 23, 15$. So $0.5X + 14 = 23$ gives $X = 18$

4 Cops and Robbers (6 points)

There are three banks called bank A, bank B and bank C. The total cash stored in these banks is 2 billion, 1 billion, and 1 billion dollars, respectively.

There is a robber who is planning to steal from one of these three banks, and there is a cop who can guard one of these three banks.

If the robber decides to steal from bank A and the cop is NOT guarding bank A, then the robber gets all of the cash in bank A. Alternatively, if the robber decides to steal from bank A, but the cop is guarding bank A, then the robber fails and gets none of the cash (and similarly for banks B and C).

1. Modelling the situation as a zero-sum game between the robber (as row player) and cop (as column player), what is the payoff matrix?

2. Write the linear program for the optimal strategy for the robber.

Solution:

Payoff matrix:

0	2	2
1	0	1
1	1	0

LP:

$$\begin{aligned} \text{Maximize} \quad & z \\ \text{Subject to} \quad & x_2 + x_3 \geq z \\ & 2x_1 + x_3 \geq z \\ & 2x_1 + x_2 \geq z \\ & x_1, x_2, x_3 \geq 0 \\ & x_1 + x_2 + x_3 = 0 \end{aligned}$$

5 Set Cover (5 points)

Alice executes the greedy algorithm for set cover on an instance for which the size of the universe is 1000. The optimal solution to the instance consists of 6 sets.

The greedy algorithm produces a set cover consisting of 20 sets. Let S_i denote the i^{th} set chosen by the greedy algorithm.

Let us suppose $|S_1 \cup S_2| = 400$.

State and prove a lower bound for $|S_3|$. (Full points for the best possible lower bound)

Claim: $|S_3| \quad \geq$ [ ]

Proof of claim:

100. If both S_1 and S_2 are both not in optimal, then optimal's six sets cover all 600 remaining items. Therefore, one of them covers at least 100 items. Then the next set selected by greedy must cover at least 100 new items.

Looser lower bounds are $\frac{1000}{20} = 50$ and $\lceil \frac{600}{18} \rceil = 34$ by a similar argument with the collection of sets taken by greedy.

6 Multiplicative Weights Update (6 points)

Alice and Bob are playing the rock-paper-scissors game. Here is the payoff matrix for the game.

	Rock	Paper	Scissor
Rock	0	-1	1
Paper	1	0	-1
Scissor	-1	1	0

Table 1: Payoff matrix of rock-paper-scissors

Alice and Bob play the game 10^{20} times with each other.

In these games, Bob ends up playing Rock 5×10^{19} times, Paper 2×10^{19} and Scissors 3×10^{19} times (in some unknown order).

1. What is the best fixed move that Alice could have played, in hindsight, given how Bob played?

   ☐ Rock
   ☐ Paper
   ☐ Scissors

2. Suppose Alice had used the multiplicative weights update algorithm to decide on her move each round. What would the best possible choice for the parameter ϵ in the multiplicative weights algorithm be? (Estimate the value of ϵ approximately, within factor of 10. You may estimate $\log 3 = 1$)

3. Suppose Alice uses the multiplicative weights algorithm with the best value of ϵ, what is the minimum possible value of her total score? (Estimate total score approximately, within a 1% error)

(You can show your work in the box below for partial credit)

Solution:

1) Paper, as Bob plays rock the most.

2) $\epsilon = \sqrt{\frac{\ln n}{T}} \approx 10^{-10}$ (from optimizing the $\epsilon T + \ln n/\epsilon$ bound)

3) The score of the best strategy is $5 * 10^{19} - 3 * 10^{19} = 2 * 10^{19}$. The regret of multiplicative weights is bounded by $O(\sqrt{\log n \cdot T}) < 4 * 10^{10}$. (Plug in ϵ to $\epsilon T + \frac{\log n}{\epsilon}$, then scale up by 2 since losses are in the range $[-1, 1]$).

   So the total score is at least $2 * 10^{19} - 4 * 10^{10}$.

7 Running a Gym (12 points)

You are running a gym and need to hire trainers for it. Here are the constraints:

- There are D days in total.
- There are n instructors for hire. The i^{th} instructor is available for hire from day a_i to day b_i. The numbers $\{a_i, b_i\}$ for $i = 1 \ldots n$ are given as input.
- Each instructor can be hired for their entire period of availability, i.e. if we hire the i^{th} instructor, then they would be at the gym every day from day a_i to day b_i.
- There should be **exactly** k instructors at the gym every day.

Your goal is to determine which instructors to hire so that the above constraints are met.

Devise an efficient algorithm that solves the problem by constructing a directed graph G with a source s and sink t, and computing the maximum flow from s to t.

1. How many nodes does G have?
2. Describe the nodes of the graph G.
3. Describe the edges of the graph G (draw a picture if needed).

Solution:

1. $D+2$ or $n+2$.

2. $D+2$ **vertex solution:** We create one vertex for each day $d = 1 \dots D$. We also create a start node s and an end node t, which we will respectively index as 0 and $D+1$ for simplicity of the arguments below.

   $n+2$ **vertex solution:** One vertex for each instructor $i = 1 \dots n$, plus s and t.

3. $D+2$ **vertex solution:** Create an edge $u \to v$ for $u, v \in 0, \dots, D$ if there is some instructor who is available from day $v+1$ to day u, and set its weight to the number of instructors with this availability range. Note that we take vertex s to have index 0 for the purpose of these calculations. Additionally, add an edge from vertex D to the end node t with weight k.

   Proof of correctness (not needed for credit): To check whether there is a valid scheduling for a setup, check whether the max flow from s to t is equal to k. To prove this reduction is correct, we must show that this happens if and only if there is a valid scheduling.

   Valid scheduling implies max flow of k: Given a valid scheduling, construct a max flow with value k as follows. For each instructor chosen in the scheduling, add a unit of flow along the edge corresponding to their availability range. Then add k units of flow from D to t. Note that the flow into a vertex is equal to the number of chosen instructors whose range ends on that day, and (except for in the case of vertex D) the flow out is equal to the number of instructors whose range ends the next day. Because these quantities must always be equal for a valid scheduling, conservation constraints are met for vertices $1, ..., D$. The flow out of vertex D is k by construction, and the flow into it must be D as well because all valid assignments have k chosen instructors with ranges ending on the last day, so its conservation constraint is also met. The flow also clearly has value k because k units of flow enter t.

   Max flow of k implies valid scheduling: If there is a max flow with size k, then there must be an integral max flow of the same value by integrality of the capacities. Choose such a flow, and construct a valid scheduling as follows. For each unit of flow along an edge u, v, select an instructor with availability range $u+1, v$. Note that number of instructors scheduled for day d is the number of selected instructors who start on/before d and end after d, which is equal to the value of the flow across the cut separating vertices with indices less than d from those with indices at least d. The value of all such s/t cuts must be k, so there are exactly k instructors scheduled for each day.

   $n+2$ **vertex solution:** Connect vertices i and j with a unit capacity edge if $a_j = b_i + 1$. Connect all vertices with $a_i = 0$ to s and $b_i = D$ to t.

   Proof of correctness: Similar to the $D+2$ solution, paths from s to t correspond to sequences of instructors covering all d days.

   Note: The $n+2$ solution is preferable to the other solution because D is only given as a parameter (whereas we are given a list of n instructors), so the $D+2$-vertex solution is actually pseudo-polynomial time. Both were given full points, however.

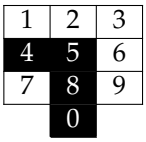
8 Phone number (15 points)

Your friend recently gave you their phone number, which you promptly forgot. While trying to remember what it was, you were able to recall two properties it satisfied:

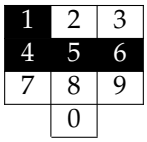
1. Its first digit was the number 0.
2. Your friend is a big fan of the game chess, and their favorite piece is the knight. As a result, each digit in their phone number can be reached from the previous digit by the move of a knight on a numpad of the following shape:

1	2	3
4	5	6
7	8	9
	0	

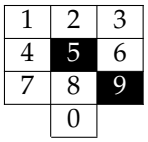
Recall that a knight is able to move in one of two ways: (i) it can move two spaces in one direction and then one space in a different direction (which is not backwards), or (ii) it can move one space in one direction and then two spaces in a different direction (which is not backwards). For example, the following three figures show three potential moves.



(a) The phone number can include the digits 40 (in that order) because 0 is reachable from 4 by a knight's move.



(b) The phone number can include the digits 61 (in that order) because 1 is reachable from 6 by a knight's move.



(c) The phone number *cannot* include the digits 59 (in that order) because 9 is *not* reachable from 5 by a knight's move.

A phone number is said to be *valid* if it satisfies these two properties. For example, 049267 is a valid phone number, but 06721 is not a valid phone number because 1 is not allowed to follow 2.

Devise a dynamic programming based algorithm for the problem of computing the number of all possible valid n-digit phone numbers.

Assume that arithmetic operations take $O(1)$ time and integers take $O(1)$ space to store.

1. What are the subproblems? (a very precise and succinct definition is needed to receive any credit)
   Solution: $\mathrm{dp}(n, l)$, n is number of digits, l is the last digit or $\mathrm{dp}(n, i)$, n is number of digits, i is the first digit

2. Describe the recurrence relation. (You don't need to write out all the cases, but try to succinctly explain the general idea)
   Solution: First, $\mathrm{dp}(m, ???) = 0$ if $m <= 0$; $\mathrm{dp}(1, 0) = 1$, $\mathrm{dp}(1, k) = 0$ if $k \neq 0$.
   Then,
   $$\mathrm{dp}(n, l) = \sum_{l \text{ reachable from } l'} \mathrm{dp}(n-1, l')$$

get $\sum_{i=0}^{9} \mathrm{dp}(n,i)$, OR First, $\mathrm{dp}(1,???) = 1$
Then,

$$\mathrm{dp}(n,i) = \sum_{i' \text{ reachable from } i} \mathrm{dp}(n-1,i)$$

get $\mathrm{dp}(n,0)$

3. Describe how the algorithm can be implemented so as to use only $O(1)$-space.
Solution: To calculate $\mathrm{dp}(n,l)$ you only need $\mathrm{dp}(n-1,l')$ and $l' \in \{0,1,2,3,4,5,6,7,8,9\}$.

9 Shattering a tree

Consider the following computational problem.

Input: A tree T with $n+1$ nodes and a weight $W[a,b]$ for each edge (a,b) in the tree.

Definition (k-shattering set): *A subset of edges E^* of the tree T is called a "k-shattering set" if deleting E^* breaks the tree into connected components each of which has at most k nodes.*

Output: Find the k-shattering set of lowest total weight.

We will devise dynamic programming based algorithms for special cases of this problem.

9.1 Shattering a path (15 points)

Assume that the tree T is just a path on $n+1$ nodes, and let W_i denote the weight of the i^{th} edge on the path. Devise a dynamic programming based algorithm for finding the lowest weight of a k-shattering set. (the runtime of the algorithm should be a polynomial in n, k)

1. What are the subproblems? (precise and succinct definition needed)

2. What is the recurrence relation?

3. What is the runtime of the algorithm?

Solution 1

1. $F[i]$ = minimum total weight of a k-shattering for the tree consisting of the first i nodes along the path.

2. Base Case: $F[i] = 0$ for all $i \leq k$
3. To write a recurrence relation for $F[i]$: Suppose we shatter the tree consisting of first i nodes. Consider the connected component containing the node i in the shattering. This connected component consists of the last ℓ nodes along the path, for some ℓ between 1 and k. So we can write a recurrence relation:

$$F[i] = \min_{\ell=1,\ldots,k} F[i-\ell] + W[i-\ell, i-\ell+1]$$

4. $O(nk)$ since there are $O(n)$ subproblems, each taking $O(k)$-time to compute.

Solution 2

1. $f(i,j)$ = minimum cost of a k-shattering set for the nodes in the range i to j inclusive.
2. Base cases: $f(i,j) = 0$ for $j - i \leq k$
   recurrence: $f(i,j) = \min_{i \leq m < j}(f(i,m) + f(m+1,j) + w[m][m+1])$
   Proof sketch: the base cases are trivially correct, since any continuous segment less than k nodes don't need to remove any edges. Assume that all sub-problems for intervals of length l or less is correctly calculated. For a length $l+1 > k$ interval (i,j) we need to delete at least one edge to get a k-shattering set. Thus, we simply try individually deleting each edge in that range. Each time an edge is deleted, the range gets split into two smaller ranges for which we can individually calculate the cheapest k-shattering set. Thus, the total cost is just the minimum over the sum of cost of deleting the edge and the cost of the cheapest k-shattering edge to the left and right of the deleted edge.
3. $O(n^3)$ since there are $O(n^2)$ subproblems, each of which takes $O(n)$ to compute (specifically, the runtime for $n+1$ nodes is equal to the following sum $\Sigma_{i=0}^{n} i * (n-i)$).

9.2 Shattering a binary tree (20 points)

Assume that the tree T is a binary tree, where each non-leaf node has exactly two children. Let $W[a,b]$ denote the weight of an edge (a,b) in the tree. Devise a dynamic programming based algorithm for finding the lowest weight of a k-shattering set. (the runtime of the algorithm should be a polynomial in n,k)

1. What are the subproblems? (precise and succinct definition needed)

2. What is the recurrence relation?

1. For each node u in the binary tree and each $\ell \in \{1,\ldots,k\}$, define $F(u,\ell)$ = minimum cost of a k-shattering set for the subtree rooted at u where the connected component containing node u has ℓ nodes.
2. Base case: $F(u,1) = 0$ where u is a leaf.

Now we will write the recurrence relation. Consider a node u and $\ell \in \{1, \ldots, k\}$. Let u_L and u_R be the left and right child of the node u respectively

$$F(u,1) = W[u,u_L] + W[u,u_R] + \min_{a,b \in \{1,\ldots,k\}} \{F(u_L,a) + F(u_R,b)\}$$

$$F(u,i) = \min \begin{cases} F(u_R,i-1) + F(u_L,a) + W[u,u_L] & \forall a \leq k-1 \\ F(u_R,i-1) + F(u_L,a) + W[u,u_R] & \forall a \leq k-1 \\ F(u_L,a) + F(u_R,i-a-1) & \forall a \leq i-1 \end{cases}$$

Proof sketch: Let us figure out the minimum cost of a k-shattering set for the subtree rooted at node u. There are four options:

(a) Case 1: remove both the left and right edges

(b) Case 2: only remove the left edge

(c) Case 3: only remove the right edge

(d) Case 4: remove neither edges.

Case 1 is a special case where you only have one node in the connected component containing i. In this case, you add the cost of deleting both edges and the minimum cost of the k-shattering set of the left subtree and of the right subtree. For case 2, you need to add the cost of the left edge and you need the right subtree to have a connected component of size $i-1$ in order for the component containing u to be size i. Then just add the minimum cost of the k-shattering set for the left subtree. Case 3 works by symmetry. Case 4 just appends u to the connected components containing it's left and right children, so you minimize the cost of the k-shattering set for both subtrees such that the number of nodes in your connected component containing u sum up to i.

3. $O(nk^2)$ since there are $O(nk)$ subproblems, each of which takes $O(k)$ to compute. There are also $O(n)$ subproblems under case 1, each of which takes $O(k^2)$.

9.3 Shattering an arbitrary tree

(5 points only, attempt after everything else)

Suppose T is an arbitrary tree and $W[a,b]$ denotes the weight of edge (a,b) in the tree. Devise a dynamic programming based algorithm for finding the lowest weight of a k-shattering set. (the runtime of the algorithm should be a polynomial in n,k)

1. What are the subproblems?

2. What is the recurrence relation?

Let us root the tree at an arbitrary node, so we have a rooted tree. Hence for each node u, let T_u denote the subtree under the vertex u (i.e., subtree rooted at u), C_u denote the set of children of u, and let $c_u = |C_u|$

Further, for each node u, let us arbitrarily order its children. For a node u, let $C(u)$ denote the set of its children. Suppose $C(u) = \{c_1, \ldots, c_\ell\}$, define trees $T(u,1), T(u,2), \ldots, T(u,\ell)$ as follows,

$$T(u,r) = \text{ subtree consisting of vertex} u \text{ and subtrees of the first } r \text{ children of} u$$

Now we are ready to define the subproblems:

$F[u,r,\ell]$ = minimum total weight of a shattering set of $T[u,r]$ where the vertex u is in a connected component of size ℓ

Let us write out the base cases, just so we understand the subproblems better.

$$F[u,r,1] = \sum_{v \in \text{ first } r \text{ children of } u} W[u,v]$$

Now to write the recurrence relation for the problem. Consider the tree $T[u,r]$. Suppose S is the optimal shattering set for $T[u,r]$ among those in which u is in a connected component of at most ℓ vertices.

Let v be the r^{th} child of u. The subtree T_v rooted at v. The connected component of vertex u can have $0,1,2,\ldots,\ell-1$ vertices inside T_v. Thus we can write the recurrence:

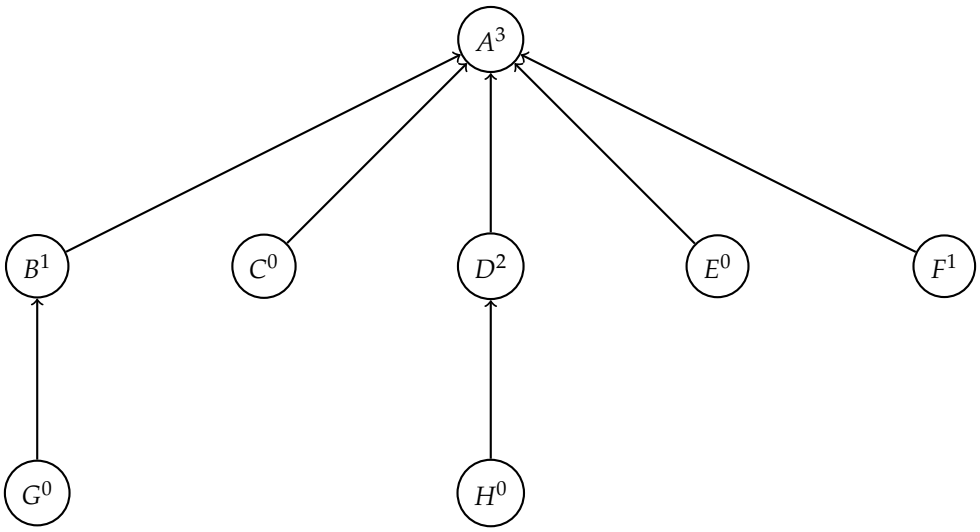
$$F[u,r,\ell] = \min \begin{cases} \min_{m=1,\ldots,\ell-1} F[u,r-1,\ell-m] + F[v,c_v,m] \\ F[u,r-1,\ell] + W[u,v] + \min_{s=1,\ldots,k}\{F[v,c_v,s]\} \end{cases}$$

10 Union find with path compression (7 points)

Recall the union find data structure which we implemented via a disjoint-set forest using path compression. Consider the following sequence of operations.

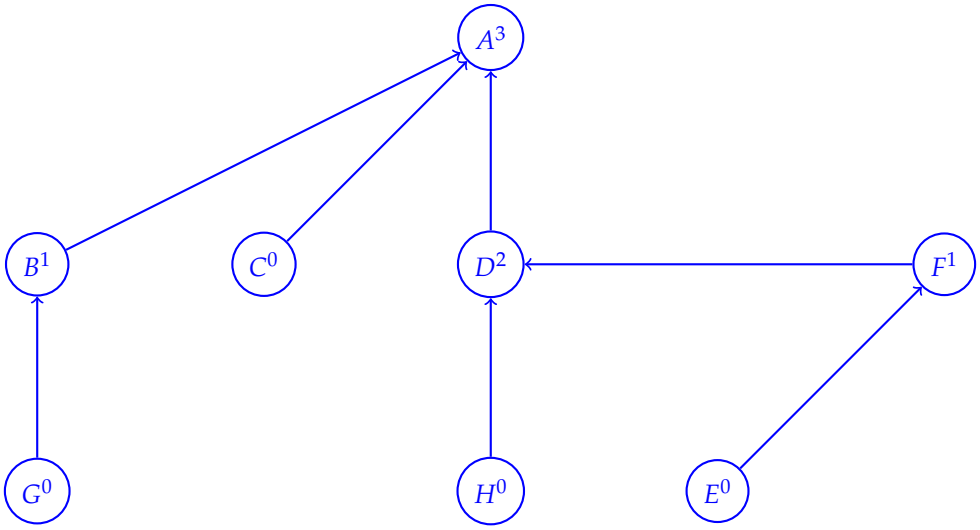
1. First, we call makeset(x) once on each letter in $\mathcal{L} = \{A, B, C, D, E, F, G, H\}$.
2. Next, we perform a sequence of union(x, y)'s, where at each step both x and y are the roots of their respective sets.
3. Finally, we perform a single find(x), on a letter $x \in \mathcal{L}$ which may or may not be the root of its set.

Suppose after these operations are complete, the data structure looks as follows:



In this diagram, x^i means that the node contains the letter x and has rank i.

1. Draw what the directed tree looked like before the find(x) in step 3.



2. For which letter $x \in \mathcal{L}$ was find(x) called in step 3?

E

Note that C and E are symmetrical and so can be exchanged in both parts, so long as this is done consistently.

Blank scratch page.