Final

Name: Targaryen

SID: 0123456789

Name and SID of student to your left: Lannister

Name and SID of student to your right: Stark

Exam Room:

☐ Wheeler 150 ☐ Haas Faculty Wing F295 ☐ Lewis 100 ☐ McCone 141 ☐ Evans 100
☐ Wozniak Lounge ☐ other

Please color the checkbox completely. Do not just tick or cross the box.

Rules and Guidelines

- **The exam is out of 170 points and will last 170 minutes.** Roughly, one should expect to spend a minute for a point.
- 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.
- Good luck!

Discussion Section

Which of these do you consider to be your primary discussion section(s)? 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.**

- ☐ Antares, Tuesday 5 - 6 pm, Mulford 240
- ☐ Kush, Tuesday 5 - 6 pm, Wheeler 224
- ☐ Arpita, Wednesday 9 - 10 am, Evans 3
- ☐ Dee, Wednesday 9 - 10 am, Wheeler 200
- ☐ Gillian, Wednesday 9 - 10 am, Wheeler 220
- ☐ Jiazheng, Wednesday 11 - 12 am, Cory 241
- ☐ Sean, Wednesday 11 - 12 am, Wurster 101
- ☐ Tarun, Wednesday 12 - 1 pm, Soda 310
- ☐ Jerry, Wednesday 1 - 2 pm, Wurster 101
- ☐ Jierui, Wednesday 1 - 2 pm, Etcheverry 3113
- ☐ Max, Wednesday 1 - 2 pm, Etcheverry 3105
- ☐ James, Wednesday 2 - 4 pm, Dwinelle 79
- ☐ David, Wednesday 2 - 3 pm, Barrows 140
- ☐ Vinay, Wednesday 2 - 3 pm, Wheeler 120
- ☐ Julia, Wednesday 3 - 4 pm, Wheeler 24
- ☐ Nate , Wednesday 3 - 4 pm, Evans 9
- ☐ Vishnu, Wednesday 3 - 4 pm, Moffitt 106
- ☐ Ajay, Wednesday 4 - 5 pm, Hearst Mining 310
- ☐ Zheng, Wednesday 5 - 6 pm, Wheeler 200
- ☐ Neha, Thursday 11 - 12 am, Barrows 140
- ☐ Fotis, Thursday 12 - 1 pm, Dwinelle 259
- ☐ Yeshwanth, Thursday 1 - 2 pm, Soda 310
- ☐ Matthew, Thursday 2 - 3 pm, Dwinelle 283
- ☐ Don't attend Section.

1 Zero-Sum Games (5 points)

Consider a zero-sum game given by the following matrix (indicating the payoffs to the row player). Here the row player is trying to maximize their payoff given by the following matrix.

	C_1	C_2	C_3
R_1	5	3	6
R_2	3	2	7
R_3	-1	4	-1

Assuming the row-player goes first, write a linear program to find their optimal strategy.

Solution: The row player goes first, so they submit a strategy (x_1, x_2, x_3). The column player already knows the row player strategy, so they pick which one of C_1, C_2, C_3 minimizes the row player's score, which is would be the expression

$$\min(5x_1 + 3x_2 - x_3, 3x_1 + 2x_2 + 4x_3, 6x_1 + 7x_2 - x_3)$$

The row player knows this happens, so they pick $\max \min(5x_1 + 3x_2 - x_3, 3x_1 + 2x_2 + 4x_3, 6x_1 + 7x_2 - x_3)$. To phrase this as a linear program, we can add a new variable p, which is upper bounded by all of $5x_1 + 3x_2 - x_3, 3x_1 + 2x_2 + 4x_3, 6x_1 + 7x_2 - x_3$. We try to maximize p, so then p will be the minimum of these three.

We want also that $[x_1, x_2, x_3]$ is a probability distribution.

Use $x_1, x_2, \ldots$ as variables in the LP.

1. What is the objective function of the LP?

Solution: $\max p$

2. What are the constraints of the linear program?

Solution:

$$\begin{aligned} 5x_1 + 3x_2 - x_3 &\geq p \\ 3x_1 + 2x_2 + 4x_3 &\geq p \\ 6x_1 + 7x_2 - x_3 &\geq p \\ x_1 + x_2 + x_3 &= 1 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

2 Runtime Analysis (8 points)

What is the runtime of the following piece of code? Write the recurrence.

```
Function what($n$) {
    what($n/2$)
    what($n/2$)
    $k = n$
    While ($k > 1$) {
        for $i = 1,2,3,...,k$ {
            for $j = 1,2,3,...,\frac{k}{2}$ {
                print BLAH
            }
        }
        $k = k/4$
    }
    what($n/2$)
    what($n/2$)
}
```

1. Recurrence Relation

   Solution: $T(n) = 4T(n/2) + \mathcal{O}(n^2)$.

   There are 4 calls to the recursive `what` with parameter $n/2$.

   The two inner for-loops take $\mathcal{O}(k^2)$ time. We do this for $k = n, n/4, n/16, \ldots, n/(4^{\log_4 n})$. So our total run-time is $n^2 + (n/4)^2 + \ldots + (n/(4^{\log_4 n}))^2 = n^2(1 + 1/16 + 1/16^2 + \ldots)$. This is n^2 multiplied by a geometric series, so it is $\mathcal{O}(n^2)$.

2. Runtime =

   Solution: $T(n) = \mathcal{O}(n^2 \log n)$.

   Use the master theorem:

   Thm. If $T(n) = aT(n/b) + \mathcal{O}(n^d)$ for some constants $a > 0$, $b > 1$, and $d \geq 0$, then

$$T(n) = \begin{cases} \mathcal{O}(n^d) & \text{if } d > \log_b a \\ \mathcal{O}(n^d \log n) & \text{if } d = \log_b a \\ \mathcal{O}(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

   In this case, $a = 4$, $b = 2$, $d = 2$, so $d = 2 = \log_2 4 = \log_b a$, and the third case applies, so $T(n) = \mathcal{O}(n^2 \log n)$.

   Note: Personal pet peeve of Max: There's no one called ”Master” who discovered this theorem. It's “The Master Theorem”, not “Master's theorem”.

3 Count-Min Sketch (10 points)

Here is the state of the count-min sketch data structure after it has processed a stream of items.

Hash function h_1	10	5	3	2	17	1	1	1
Hash function h_2	10	6	2	4	6	4	4	4
Hash function h_3	13	3	3	6	1	4	4	~~4~~ 6
Hash function h_4	1	2	3	4	5	6	9	10

Solution: During the exam as a correction, the last element of the row corresponding to the hash function h_3 was made into 6.

For an element A, let $trueCount(A)$ denote the total number of occurrences of A in the stream, and let $estimate(A)$ denote the estimate of the number of occurrences of A as per the count-min data structure.

1. What is the length of the stream?

   Solution: 40. One bucket for each row for each element in the stream, and every row adds up to 40.

2. What is the largest possible value of $trueCount(A)$ for an element A?

   Solution: 10. The true value for A is upper bounded by the value in the bucket it hashes to in each row. Rows 2 and 4 have 10 as their largest number, so the largest possible value is 10.

3. What is the smallest possible value of $estimate(A)$ for an element A?

   Solution: 1. *estimate* will take the minimum value for the buckets each element hashes to in each row. The smallest such value is 1.

4. For two elements A and B, what is the maximum possible value of $trueCount(A) + trueCount(B)$?

   Solution: 16. Like for 2, the true values are upper bounded by the buckets they hash to in each row. If in any row they hash to the same value, $trueCount(A) + trueCount(B)$ is upper bounded by that value, otherwise $trueCount(A) + trueCount(B)$ is upper bounded by $M[i, h_i(A)] + M[i, h_i(B)]$. The largest such value happens in the second row, with 10 and 6.

5. What is the largest possible value of $estimate(A) + estimate(B)$?

Solution: 20. For the largest possible value for $estimate(A)$, look at the largest value in each row, and then the smallest among all the rows. That would be 10. If A and B hash to the same thing in each row, then the maximum of the sum would be 20. If they hashed to different things, then the estimates could only decrease.

4 Integrality of Maximum Flow (3 points)

Every network with integer edge capacities has a maximum flow where the amount of flow on each edge is an integer. Briefly justify.

Solution: We know that the Edmonds-Karp max-flow algorithm computes a maximum flow. It starts with zero flow, so all flow values are integers. Each time we push flow along a new path, the remaining flow we can push is an integer, so the flow values remain integers. Then the max flow Edmonds-Karp computes has only integer flow values.

5 MinCut (4 points)

Let $G = (V, E)$ be a network with source s and sink t. Note that G may have more than one minimum s-t cut and more than one maximum s-t flow.

For each of the following, let B be the value of the maximum s-t flow of G. Fill the appropriate circle in each of following cases.

1. Let e be an edge across a minimum s-t cut. Suppose we decrement the capacity of edge e by 1, what is the value of the maximum s-t flow in the resulting network?

   ○ $B - 1$
   ○ B
   ○ Depends on the network.

   Solution: B - 1. This makes the minimum cut even smaller. The max-flow min-cut theorem says the minimum cut equals the max flow, so the max flow is $B - 1$.

2. Let e be an edge on one of the maximum s-t flows. Suppose we decrement the capacity of edge e by 1, what is the value of the maximum flow in the new network?

   ○ $B - 1$
   ○ B
   ○ Depends on the network.

   Solution: Depends on the network. If e is a bottleneck, then decrementing the capacity of e will decrement the max-flow. But if e is not a bottleneck, then the max-flow will stay the same when decrementing e.

6 Dynamic Programming Orders (8 points)

A dynamic programming algorithm has inputs $X[1,\ldots,n]$ and $Y[1,\ldots,n]$, subproblems $E[i,j]$ for all $i,j \in \{1,\ldots,n\}$, and the following recurrence relation,

$$E[i,j] = \min \begin{cases} E[i-1,j-1]+1 & \text{if } X[i] = Y[j-1] \\ E[i-2,\lceil j/2 \rceil]+3 & \text{if } X[i] = Y[j] \\ E[i-1,j+1]+1 & \text{if } X[i+1] = Y[j+5] \end{cases}$$

1. Fill in the blanks in the following pseudocode for the DP algorithm. (It is OK for the code to go out of bounds on E. In other words, assume that $E[i,j]$ is well-defined and already computed correctly for i, j in $\{-2,-1,0,n+1,n+2\}$)

```
for $\boxed{i}$ from 1 to $n$ do {
    for $\boxed{j}$ from 1 to $n$ do {
```

$$E[i,j] = \min \begin{cases} E[i-1,j-1]+1 & \text{if } X[i] = Y[j-1] \\ E[i-2,j/2]+3 & \text{if } X[i] = Y[j] \\ E[i-1,2j+5]+1 & \text{if } X[i+1] = Y[j+5] \end{cases}$$

```
    }
}
```

Solution: We must put i and j in these boxes because they are used to compute the recurrence. It cannot be j first, then i, because $E[i,j]$ depends on both $j+1$ and $j-1$.

2. Suppose our goal is to compute $E[n,n]$ in polynomial time. What is the smallest memory with which you can implement the above algorithm?

$\boxed{\mathcal{O}(n)}$

Solution: Let row 1 of E be $E[1,1],\ldots,E[1,n]$, row 2 be $E[2,1],\ldots,E[2,n]$, ..., row n be $E[n,1],\ldots,E[n,n]$.

To compute the current row i, we need row $i-1$ and $i-2$, so we store the previous two rows in addition to the current one. This is $\mathcal{O}(n)$.

$2n$ and $3n$ were also accepted as answers.

Describe your implementation in a couple of sentences.

Solution: We compute using the recurrence and the same order as above, except we keep the previous two rows in memory while computing $E[i,j]$ for the current one, rather than keeping the entire table in memory.

7 NP-completness true/false (10 points)

For each of the following questions, there are four options:

(1) True (T); (2) False (F); (3) True if and only if $\mathbf{P} = \mathbf{NP}$; (4) True if and only if $\mathbf{P} \neq \mathbf{NP}$.

Circle one for each question.

Note: By "reduction" in this exam it is always meant "polynomial-time reduction with one call to the problem being reduced to."

1. There is a reduction from Independent Set to the Longest Increasing Subsequence problem.

   ○ T ○ F ○ $\mathbf{P} = \mathbf{NP}$ ○ $\mathbf{P} \neq \mathbf{NP}$

   Solution: P = NP. If such a reduction existed, it would give a polynomial-time algorithm for Independent Set. Because Independent Set is NP-complete, this means that there's a polynomial time for every NP problem. If P=NP, then there's a trivial reduction from Independent Set to LIS using the polynomial-time solution algorithm for Independent Set.

2. Every problem in P reduces to the 3-SAT problem.

   ○ T ○ F ○ $\mathbf{P} = \mathbf{NP}$ ○ $\mathbf{P} \neq \mathbf{NP}$

   Solution: True. 3-SAT is NP-complete, $P \subseteq NP$, so every problem in P reduces to 3-SAT.

3. Every problem in NP reduces to the 3-SAT problem.

   ○ T ○ F ○ $\mathbf{P} = \mathbf{NP}$ ○ $\mathbf{P} \neq \mathbf{NP}$

   Solution: True. This is the definition of NP-hard, and 3-SAT is NP-complete (and so NP-hard).

4. 3-SAT problem reduces to every NP-complete problem.

   ○ T ○ F ○ $\mathbf{P} = \mathbf{NP}$ ○ $\mathbf{P} \neq \mathbf{NP}$

   Solution: True. 3-SAT is NP, and any problem in NP reduces to any NP-complete problem.

5. The 3-SAT problem reduces to some problem in P.

   ○ T ○ F ○ $\mathbf{P} = \mathbf{NP}$ ○ $\mathbf{P} \neq \mathbf{NP}$

   Solution: P=NP. This is exactly like in 1. If such a reduction existed, it would give a polynomial-time algorithm for 3-SAT, and therefore for any NP problem. If P=NP, then there's a trivial reduction from 3-SAT to some P problem using the polynomial-time solution algorithm for 3-SAT.

8 True/False. (14 pts)

1. The worst case complexity of the simplex algorithm for linear programming is exponential, but linear programming is in P.

   ◯ True ◯ False

   Solution: True. The simplex algorithm is exponential time in the worst case, but there are algorithms (e.g. ellipsoid) that solve LPs in polynomial time.

   As an interesting side note, it was observed that the concrete performance of simplex was quite good, and it outperformed polynomial-time algorithms. This helped motivated techniques beyond worst case analysis, such as average-case analysis.

2. For every directed graph, there exists a starting point v, such that calling the $explore(v)$ routine in DFS will visit every node in the graph.

   ◯ True ◯ False

   Solution: False. If there is some pair of nodes u, v where there is no path from u to v and none from v to u, then an explore from one will never reach the other.

3. For every directed graph and for each SCC in the graph, there exists a starting point v, such that calling the $explore(v)$ routine will visit every node in that SCC of the graph.

   ◯ True ◯ False

   Solution: True. Pick any node in the SCC, a single $explore$ call will visit every node in the SCC.

4. For a graph G, there is always some MST of G that does not contain the heaviest edge.

   ◯ True ◯ False

   Solution: False. If G is just a tree, then there's only one MST, and it will not avoid the heaviest edge.

5. Suppose u, v are children of the root in a BFS search tree of a connected undirected graph. Then deleting the root will always disconnect u and v in the original graph.

   ◯ True ◯ False

   Solution: False. As an example, running BFS on a complete graph on n vertices will be a depth-1 tree, but removing the root will just be the complete graph on $n - 1$ vertices.

6. Suppose u, v are children of the root in a DFS search tree of a connected undirected graph. Then deleting the root will always disconnect u and v in the original graph.

   ◯ True ◯ False

   Solution: True. If u and v were connected in the original graph, then $explore(u)$ will eventually visit u, or $explore(v)$ would eventually visit u, so one must be a descendant of the other. The only case when u and v are both children of the root on the same level of a DFS tree is if there's no path between them that doesn't include the root.

7. If there is a polynomial time algorithm for 3-SAT then there is a polynomial time algorithm to factor n bit numbers.

   ◯ True ◯ False

   Solution: True. Factoring is in NP (it takes polynomial time to verify a factorization of a number is correct), so if 3-SAT can be solved in polynomial time, so can factoring.

9 Fill in the Blanks (24 points)

When asked for a bound, always give the tightest exact bound possible, not an asymptotic one. Some questions have choices in parentheses after the answer box.

1. Dr. Hurry is an impatient man. He is reading the weights on the edges of a graph G one at a time. After reading just k edges, Dr. Hurry exclaims that an edge e is not part of an MST of G. What is the smallest possible value of k?

   Solution: 3. It was clarified during the exam that all graphs are simple. If I see just one edge, then I can conclude nothing. If I see two edges, I still can't conclude anything. But if I see three edges, then one of them can be the heaviest in a cycle, and then wouldn't be in any MST.

2. Let us suppose we execute the Follow the regularized leader (FTRL) algorithm over convex set $[-1, 1]$, with regularizer $R(x) = x^2$. Suppose the cost functions in first three rounds are $f_1(x) = 1 + x$, $f_2(x) = 1 - x$ and $f_3(x) = 1 + x$. Then, the value of x suggested by FTRL algorithm for the fourth step is $= \arg\min_{x\in[-1,1]}($ $)$.

   Solution: $3 + x + x^2$ Formally, to find $x^{(4)}$ FTRL picks

$$
\begin{aligned}
x^{(4)} &= \arg\min_{x\in[-1,1]} R(x) + \sum_{i=1}^{3} f_i(x) \\
&= 3 + x + x^2
\end{aligned}
$$

3. Setting the regularizer function to be a constant function $R(x) = 10$ in any FTRL algorithm, we recover the algorithm.

   Solution: Follow-The-Leader. Setting the regularizer to be a constant doesn't bias toward any x, which is the behavior of follow-the-leader.

4. If a graph G has some vertex cover of size k, the size of the maximum matching is at most

   Solution: k. Size of a matching is the number of edges. Each edge in the matching must be vertex-disjoint. If there's a vertex cover of size k, then if we tried have a $k + 1$-edge matching by the pigeonhole principle some vertex in this cover is incident to at least two edges in this matching.

5. If a graph G has a maximum matching of size k, the size of the minimum vertex cover is at most

Solution: $2k$. Assume the vertex cover picked some vertex v not covered by the maximum matching M. v must be incident to some edge (if not, the vertex cover could be smaller by not picking it). If this vertex has some edge to a vertex not covered by M, this edge can be added to M to make it bigger.

So v can only have edges to vertices covered by M. But then a vertex cover would be all the vertices covered by M (this is $2k$), so the minimum vertex cover is at most $2k$ vertices.

6. Suppose we draw a hash function $h : \mathbb{Z}_p \to \mathbb{Z}_p$ from a universal/pairwise independent hash family $\mathcal{H}$, then the probability that $h(100) = h(10)^2 + 5 \mod p$ is at most

Solution: $\frac{1}{p}$.

We want to find $P(h(100) = h(10)^2 + 5 \pmod p)$. We can decompose this using the total probability rule:

$$\begin{aligned} P(h(100) = h(10)^2 + 5 \pmod p) &= \sum_{a \in B} P(h(100) = h(10)^2 + 5 \pmod p \mid h(10)^2 = a) P(h(10)^2 = a) \\ &= \sum_{a \in B} \frac{1}{p} P(h(10)^2 = a) \\ &= \frac{1}{p} \sum_{a \in B} P(h(10)^2 = a) \\ &= \frac{1}{p} \end{aligned}$$

$P(h(100) = h(10)^2 + 5 \pmod p \mid h(10)^2 = a) = \frac{1}{p}$ by pairwise independence. $\sum_{a \in B} P(h(10)^2 = a) = 1$ as a sum of probabilities of disjoint events.

7. The greedy algorithm for HornSAT returns the assignment 00011 on a formula Φ with 5 variables. What is the maximum number of satisfying assignments that formula can have?

Solution: 8. HornSAT makes true only what *must* be true. If there are no other constraints on the variables, then any assignment of the form $b_1 b_2 b_3 11$ where $b_1, b_2, b_3 \in 0, 1$ is a satisfying assignment, and there are $2^3 = 8$ such assignments.

8. A directed graph has a cycle if and only if every depth-first search on the graph reveals a

(tree/forward/back/cross) edge.

Solution: Back. If the graph has a cycle, any DFS will find a back-edge, and there is a back-edge in some DFS there is a cycle in the graph. The same does not happen with any other type of edge.

10 Better-Than-Most TSP Tour (10 points)

An instance of TSP consists of n cities and distances $d[\cdot,\cdot]$ between every pair of cities. The distances may not satisfy the triangle inequality. A TSP tour is a path that visits every city exactly once.

There are $(n-1)!$ possible TSP tours in any instance with n cities. Finding the TSP tour that has the smallest total length among all these $(n-1)!$ tours is NP-hard. For distances that don't satisfy the triangle inequality, there are no approximation algorithms for the problem either.

Let us say that a TSP tour is *Better-Than-Most* if its cost is smaller than 99% of the $(n-1)!$ possible TSP tours.

1. Describe an algorithm that given δ in $(0,1)$, runs in polynomial-time in n and $1/\delta$, and outputs some tour which is a *Better-Than-Most* TSP tour with probability $1-\delta$.

   (*Hint*: Given two tours, comparing their costs takes linear time.)

   Solution: Sample some number, t, of TSP tours uniformly at random. Find the lowest-cost TSP tour among the t random tours with a linear scan, and return this one. The main question is how many samples t to take.

   We will return a Better-Than-Most tour with this algorithm if and only if any of our sampled tours is Better-Than-Most. The probability a randomly sampled tour is Better-Than-Most is 0.01. Let $B_1,\ldots,B_t$ be the events corresponding to B_i meaning the ith sampled tour is Better-Than-Most. Then

$$\begin{aligned} P(\text{any tour Better-Than-Most}) &= P(B_1 \cup \ldots \cup B_t) \\ &= 1 - P(\overline{B}_1 \cap \ldots \cap \overline{B}_t) \\ &= 1 - (0.99)^t \\ &\geq 1-\delta \end{aligned}$$

   Since we want $1-0.99^t \geq 1-\delta$, this means we must have $\delta \geq 0.99^t$, so $t\log(1/0.99) \geq \log(1/\delta)$. i.e. $t = \Theta(\log(1/\delta))$.

2. What is the runtime of your algorithm in terms of n and δ?

   Solution: There are a few possible answers:

   (a) Assuming sampling a random integer in any range takes constant time, we can sample a random tour in $\mathcal{O}(n)$ time. It takes $\mathcal{O}(n)$ time to compare two tours, and we do this $t = \mathcal{O}(\log(1/\delta))$ times. So our total time is $\mathcal{O}(n\log(1/\delta))$. We'd also get this time if we just ignored how long it would take to sample the tours.

(b) Assuming sampling a random bit takes constant time (i.e. an integer only in the range $\{0, 1\}$), to sample a random tour would take $\log(n-1)! = \mathcal{O}(n \log n)$ time. If we do this $\mathcal{O}(\log(1/\delta))$ times, the total time to sample all tours would be $\mathcal{O}(n \log n \log(1/\delta))$ times, and since this is more than $\mathcal{O}(n \log(1/\delta))$, our total time is $\mathcal{O}(n \log n \log(1/\delta))$.

3. Prove that the algorithm outputs a *Better-Than-Most* TSP tour with probability at least $1 - \delta$.

Solution: This is basically the analysis we did in 1. We will return a Better-Than-Most tour with this algorithm if and only if any of our sampled tours is Better-Than-Most.

Then

$$\begin{aligned} P(\text{any tour Better-Than-Most}) &= P(B_1 \cup \ldots \cup B_t) \\ &= 1 - P(\overline{B}_1 \cap \ldots \cap \overline{B}_t) \\ &= 1 - (0.99)^t \\ &\geq 1 - \delta \end{aligned}$$

If $t = \Theta(\log(1/\delta))$, then we will output a Better-Than-Most tour with probability at least $1 - \delta$.

11 Coffee Shops (20 points)

A rectangular city is divided into a grid of $m \times n$ blocks. You would like to setup coffee shops so that for every block in the city, either there is a coffee shop within the block or there is one in a neighboring block. (There are up to 4 neighboring blocks for every block).

It costs r_{ij} to rent space for a coffee shop in block ij.

1. Write an integer linear program to determine which blocks to setup the coffee shops at, so as to minimize the total rental costs.

   (a) What are your variables, and what do they mean?

Variables	What does the variable mean?

   Solution: There is a variable for every block x_{ij}, i.e., $\{x_{ij} | i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}\}$. This variable corresponds to whether we put a coffee shop at that block or not.

   (b) What is the objective function?

   Solution: $\min \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} x_{ij}$. Alternatively, $\min t$ is correct as well as long as the correct constraint is added.

   (c) What are the constraints?

Constraint	What does the constraint enforce?

   Solution:

(a) $x_{ij} \geq 0$: This constraint just corresponds to saying that there either is or isn't a coffee shop at any block. $x_{ij} \in \{0, 1\}$ or $x_{ij} \in \mathbb{Z}_+$ is also correct.

(b) For every $1 \leq i \leq m, 1 \leq j \leq n$:

$$x_{ij} + x_{(i+1),j}\mathbb{1}_{\{i+1\leq m\}} + x_{(i-1),j}\mathbb{1}_{\{i-1\geq 1\}} + x_{i,(j+1)}\mathbb{1}_{\{j+1\leq n\}} + x_{i,(j-1)}\mathbb{1}_{\{j-1\geq 1\}} \geq 1$$

This constraint corresponds to that for every block, there needs to be a coffee shop at that block or a neighboring block.

$\mathbb{1}_{\{i+1\leq m\}}$ means "1 if $\{i + 1 \leq m\}$, and 0 otherwise". It keeps track of the fact that we may not have all 4 neighbors on the edges, for instance.

(c) If the objective was min t, then the constraint $\sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij}x_{ij} \leq t$ needs to be added.

2. Solving the linear program gets you a real valued solution. How would you round the LP solution to obtain an integer solution to the problem? Describe the algorithm in at most two sentences.

Solution: Round to 1 all variables which are greater than or equal to $1/5$. Otherwise, round to 0.

In other words, put a coffee shop on (i, j) iff $x_{i,j} \geq 0.2$.

3. What is the approximation ratio obtained by your algorithm?

Solution: Using the rounding scheme in the previous part gives a 5-approximation.

4. Briefly justify the approximation ratio.

Solution: Notice that every constraint has at most 5 variables. So for every constraint, there exists at least one variable in the constraint which has value $\geq 1/5$ (not everyone is below average). The number of coffee shops in the rounded solution is at most $5 \cdot$ LP-OPT, (the number of $x_{i,j} \geq 0.2$ is at most $5 \cdot \sum_{i,j} x_{i,j}$). Since Integral-OPT $\geq$ LP-OPT (the LP is more general than the ILP), our rounding gives value at most 5 LP-OPT $\leq$ 5 Integral-OPT. So we get a 5-approximation.

12 NP-Completeness Reductions (20 points)

Show that the following problems are NP-complete by providing a polynomial-time reduction. You may assume that the following problems are NP-complete: Rudrata (Hamiltonian) Path, Rudrata (Hamiltonian) Cycle, Vertex Cover, Independent Set and 3-SAT.

1. **Quarter Path**
   Input: Graph $G = (V, E)$, and vertex s
   Solution: There is a path with $n/4$ edges all of whose vertices are distinct

   Proof. Briefly argue that the Quarter Path problem is in NP

   Solution: To argue QUARTER-PATH is in NP, we only need to show it has a polynomial-time verification algorithm. Given a candidate path, it takes us polynomial time to check it is a valid path, is of length at least $n/4$, and that it never visits the same vertex twice.

   We will now use a reduction to show that the problem is NP-complete. Fill up the details in the proof below..

   We will exhibit a polynomial time reduction from the $\boxed{\text{Rudrata Path}}$ to the $\boxed{\text{Quarter Path}}$

   Given an instance Φ of the problem $\boxed{\text{Rudrata Path}}$ *we construct an instance Ψ of the problem* $\boxed{\text{Quarter Path}}$

   as follows ...

   Solution: Add $3n - 4$ vertices that are isolated from any other vertex.

   Note: we were lenient with rounding errors on this question. $3n$ was fine.

The proof that this is a valid reduction is as follows:

Solution: We need to prove both directions: a solution to Φ for Rudrata Path exists iff one exists for the reduced Quarter Path problem Ψ.

Assume that there is a solution to Rudrata Path Φ. So there is a path that visits all the vertices in the graph using $n-1$ edges, that doesn't visit any vertex more than once. If we add $3n-4$ vertices, the new graph has $n' = 4n-4$ vertices, $n'/4 = n-1$ which are covered by the existing Rudrata path. So this is a solution to the Quarter Path problem Ψ

Assume there is a solution to the Quarter Path instance Ψ. So some path visits $n'/4 = n-1$ vertices, visiting no vertex more than twice. The only non-isolated vertices are those in the original graph, so this path must be entirely in the original graph. So it forms a Rudrata path in Φ.

13 Just Repetition (22 points)

Dee is texting Matt using her faulty phone that inserts spurious characters into the message. To cope with these spurious characters, Dee repeats her message twice. We will devise an algorithm for Matt to recover Dee's message from what he receives.

Formally, call a string Y to be a *valid message* if Y consists of some string w repeated twice, i.e., Y is w concatenated with w for some string w.

Matt receives an input string $x[1,\ldots,n]$. Design an algorithm to find the minimum number of character deletions needed to make x into a *valid message*. Your algorithm should take time at most $O(n^3)$.

(*Hint:* use a DP algorithm as a subroutine to solve the problem)

1. Describe the main idea behind the algorithm in three to four sentences.

Solution: As a subroutine, we use a DP algorithm $\texttt{dist}_D(z, w)$. For two strings z and w, this computes the minimum edit distance from z to w, allowing insertions and deletions but no substitutions. This is the same distance it takes to get from z and w to a common string, deleting from z and w.

Then we return $\min_{1\le i\le n-1} \texttt{dist}_D(x[1,\ldots,i], x[i+1,\ldots,n])$.

2. What are the subproblems in the DP?

Solution: This is just like edit distance. The subproblems are $D[i,j]$, which represents the minimum distance to get from $z[1,\ldots,i]$ to $w[1,\ldots,j]$ using only insertions and deletions.

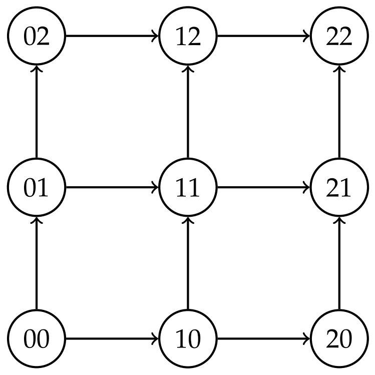
3. Write the recurrence relation.

Solution:

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1] & \text{if i = j} \end{cases}$$

14 Multiplicative Weights (12 points)

Consider the following simplified map of Berkeley. Due to traffic, the time it takes to traverse a given path can change each day. Specifically, the length of each edge in the network is a number between $[0,1]$ that changes each day. The travel time for a path on a given day is the sum of the edges along the path.



For T days, both Max and Vinay drive from node 00 to node 22.

To cope with the unpredictability of traffic, Vinay builds a time machine and travels forward in time to determine the traffic on each edge on every day. Using this information, Vinay picks the path that has the smallest total travel time over T days, and uses the same path each day.

Max wants to use the multiplicative weights update algorithm to pick a path each day. In particular, Max wants to ensure that the difference between his expected total travel time over T days and Vinay's total travel time is at most $T/10000$. Assume that Max finds out the lengths of all the edges in the network, even those he did not drive on, at the end of each day.

1. How many experts should Max use in the multiplicative weights algorithm?

   Solution: 6 experts

2. What are the experts?

   Solution: There is one expert for every path from 00 to 22. One can see that there are 6 different paths from 00 to 22.

3. Given the weights maintained by the algorithm, how does Max pick a route on any given day?

   Solution: The algorithm maintains one weight for each path. Max picks a path with probability proportional to its weight.

4. The regret bound for multiplicative weights is as follows:

   Theorem. Assuming that all losses for the n experts are in the range $[0,4]$, the worst possible regret of the multiplicative weights algorithm run for T steps is

   $$R_T \leq 8\sqrt{T \ln n}$$

   Use the regret bound to show that expected total travel time of Max is not more than $T/10000$ worse than that of Vinay for large enough T.

Solution: Let $P_1, \ldots, P_6$ be the 6 possible paths between 00 and 22. Let $\ell_i^{(t)}$ denote the length of path i on day t. Let $w_i^{(t)}$ denote the weight of path i on day t.
Since Max picks a path proportional to its weight, the expected total time on day t is

$$\sum_{i=1}^{6} w_i^{(t)} \cdot \ell_i^{(t)},$$

and the expected total time over T days is,

$$\sum_{t=1}^{T} \sum_{i=1}^{6} w_i^{(t)} \cdot \ell_i^{(t)}.$$

The regret bound to multiplicative weights asserts that for every path P_i,

$$\sum_{t=1}^{T}\sum_{i=1}^{6} w_i^{(t)} \cdot \ell_i^{(t)} - \sum_{t=1}^{T} \ell_i^{(t)} \leq 8\sqrt{T \ln 6}\,.$$

Here $n = 6$, since there are 6 different paths. Since Vinay picks one of the paths P_i to use over all days, the above regret bound implies that total expected time of Max is at most $8\sqrt{T \ln 6}$ worse than that of Vinay. For sufficiently large T, $8\sqrt{T \ln 6} < T/10000$, in particular $T > (8 \cdot 10000)^2 \cdot \ln 6$ suffices.