Simplex Algorithm
Problem Set 8 – CS6515 (Spring 2025)

- This problem set is due on **Thursday March 27th**. (This is 2 weeks, because of Spring break.)
- Submission is via Gradescope.
- Your solution must be a typed pdf (e.g., via LaTeX, Markdown, etc. Anything that allows you to type math notation) – no handwritten solutions.
- Please try to make your solutions as concise and readable as possible. Most problems will have solutions that are no more than a page long. Consider using bullet points and adding space to break up large paragraphs into smaller chunks.
- There are 3 problems. Each problem is graded with 20p. **There is +1p bonus per problem** for stating (i) how long it took you to solve that problem, and (ii) how long it took you to type the answer.

22 Faster Matrix Inversion

For a linear program $\mathbf{A}x \le b$ where $\mathbf{A} \in \mathbb{R}^{n \times d}$ $(n \ge d)$, the simplex algorithm picks d linear independent rows of $\mathbf{A}$ to form $\mathbf{M} \in \mathbb{R}^{d \times d}$. It must then compute $\mathbf{M}^{-1}$. Computing a matrix inverse from scratch takes $O(d^3)$ time[1]. However, the matrix $\mathbf{M}$ changes by only one row per iteration, so there is no need to recompute the inverse from scratch.

Problem Design and analyze an algorithm that, when given $\mathbf{M}, \mathbf{M}^{-1} \in \mathbb{R}^{d \times d}$, index $i \in \{1...d\}$, and vector $a \in \mathbb{R}^d$, returns $\mathbf{M}'^{-1}$ (where $\mathbf{M}'$ is the matrix $\mathbf{M}$ but we replaced the ith row by $a^\top$). Give algorithm description, proof of correctness, and complexity analysis.

Hint: Use the "Sherman-Morrison-Identity" which states that

$$(\mathbf{M} + uv^\top)^{-1} = \mathbf{M}^{-1} - \frac{\mathbf{M}^{-1}uv^\top\mathbf{M}^{-1}}{1 + v^\top\mathbf{M}^{-1}u}.$$

For full points, the time complexity should be $O(d^2)$.
In your algorithm description, be very clear about the order in which you multiply things.

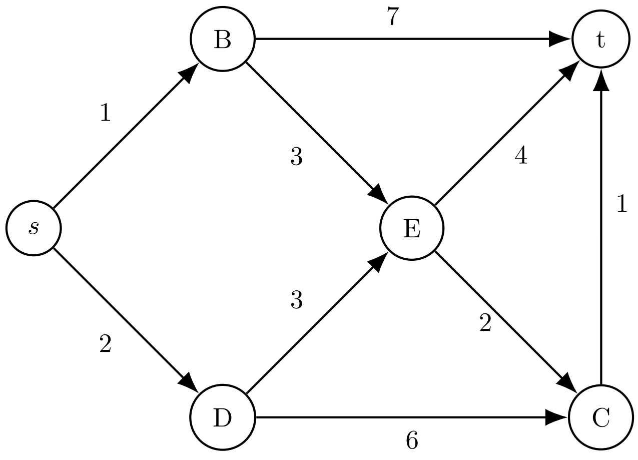
[1] $O(d^{2.372})$ is also possible via "fast matrix multiplication," but there are incredibly large constants hidden in O-notation. https://arxiv.org/pdf/2307.07970

23 Geometric Interpretation of Dijkstra's Algorithm

Let $G = (V, E)$ be a directed graph with positive edge lengths c_e. Let s, t be some vertices and we want to compute the distance from s to t. We could do that via Dijkstra's algorithm. Alternatively, we could solve the following linear program via the simplex algorithm

$$\begin{aligned} &\max_{x \in \mathbb{R}^V} x_s \\ x_u - x_v &\leq c_{uv} \quad \text{for all } (u,v) \in E \\ x_v &\geq 0 \quad \text{for all } v \in V \setminus \{t\} \\ x_t &\leq 0 \end{aligned} \tag{1}$$

In this exercise, we want to compare Dijkstra's algorithm and the simplex algorithm. Consider the following graph $G = (V, E)$:



We will use the following slightly modified Dijkstra's algorithm. The changes are highlighted (blue).

Algorithm 1: Dijkstra

```
procedure DIJKSTRA($G = (V, E), s, t$)
    Queue.insert(0, s, $\cdot$ )
    $T = \emptyset$
    while Queue not empty do
        d, $v$, e = Queue.pop() // priority queue
        if visited[$v$] then
            continue
        visited[$v$] = true
        distance[$v$] = d
        $T = T \cup \{e\}$ // Store edge used to visit $v$, i.e., $T$ is shortest path tree
        Construct a vector $x \in \mathbb{R}^V$ with $x_z$ = d − distance[$z$] for all visited $z \in V$, and $x_z = 0$ for
          unvisited $z \in V$.
        if $v == t$ then return $d, T$ // We found the distance to $t$, so we can terminate;
        for neighbor $u$ of $v$ do
            Queue.insert(d+$c_{vu}$, $u$, $(v, u)$)
```

Questions/Problems:

1. Run modified Dijkstra's algorithm on above graph. Fill the following table, each row corresponds to an iteration where we visit a vertex for the first time. List (i) the vertices that have not been visited

yet at the start of that iteration, (ii) the vertex that is newly visited, (iii) the vector x computed in that iteration, (iv) set T at the end of that iteration.
The following table shows the first 2 iterations.

Un-visited vert.	Newly visited	Vector x $\begin{bmatrix} x_s & x_B & x_C & x_D & x_E & x_t \end{bmatrix}$	tree T
s,B,C,D,E,t	s	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	$\emptyset$
B,C,D,E,t	B	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	$\{\ (s,B)\ \}$

2. Run the simplex algorithm on linear program (1) (for the same graph G as in subproblem 1.). Use initial-point $x = \vec{0}$. For each iteration list the following information: (i) x at the start of the iteration, (ii) the tight constraints, (iii) the direction in which we move next, (iv) how far we move in that direction, (v) which constraint has been removed and added to the set of tight constraints.
The following table shows the first 2 iterations.

x	tight constraints	direction	λ	tight constraint change
$\begin{bmatrix} x_s \\ x_B \\ x_C \\ x_D \\ x_E \\ x_t \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{array}{l} x_s \geq 0 \\ x_B \geq 0 \\ x_C \geq 0 \\ x_D \geq 0 \\ x_E \geq 0 \\ x_t \leq 0 \end{array}$	$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	1	Removed Constraint $x_s \geq 0$ Added Constraint $x_s - x_B \leq 1$
$\begin{bmatrix} x_s \\ x_B \\ x_C \\ x_D \\ x_E \\ x_t \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{array}{l} x_s - x_B \leq 1 \\ x_B \geq 0 \\ x_C \geq 0 \\ x_D \geq 0 \\ x_E \geq 0 \\ x_t \leq 0 \end{array}$	$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	1	Removed Constraint $x_B \geq 0$ Added Constraint $x_s - x_D \leq 2$

3. Compare the two algorithms: How do x, tight constraints, removed&added constraint in each iteration of the simplex algorithm relate to the values computed in each iteration of Dijkstra's algorithm?

Remark: You do **not** need to solve all the linear systems by hand and show calculation. Instead, it's fine if you guess+verify the direction in each iteration (i.e., check that moving x along that direction will replace only one of the tight constraints).

You do not need to write your values in a table. You can also list them in some other form, eg bullet points. Important is that you provide all the required information that is also listed in the above examples.

Context: In class, we motivated the simplex algorithm as a generalization of augmenting paths for bipartite matching. Here we see that it's also a generalization of Dijkstra's algorithm. The main difference is that the calculations are done via Linear Algebra/Geometry. The simplex algorithm also generalizes many other combinatorial optimization algorithms.

In general, the simplex algorithm has exponential worst-case time complexity, because it could have exponentially many iterations. However, in practice it often terminates after only $O(n)$ iterations. Especially for many combinatorial problems (matching, paths, flows, etc) it has few iterations since it is a geometric/algebraic interpretation of other combinatorial/graph-based algorithms.

24 Constructing the Polytope Graph

We are given a polytope via the constraints $\mathbf{A}x \leq b$ where $\mathbf{A} \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^n$, $n \geq d$. We want to compute the polytope graph (i.e., construct an undirected graph $G = (V, E)$ where each $v \in V$ corresponds to a vertex/corner of the polytope, and each edge $e \in E$ of the graph is an edge of the polytope connecting two vertices/corners).

Problem:

- Design an algorithm that, given $\mathbf{A}, b$, returns the polytope graph. (Please explain what the algorithm does and why it works. The explanation itself is not graded, but if the TA does not understand your algorithm/solution, then they can deduct points.)
- Analyze the algorithm's time complexity.

For full points, the algorithm should run in time $O(T \operatorname{poly}(n))$ where T is the number of vertices. In particular, when the number of vertices is small (i.e., bounded by $\operatorname{poly}(d)$) then the algorithm should run in polynomial time. Brute-force by trying all d-sized subsets of n constraints would be exponential in d, and not accepted.

You can assume calculation of an $n \times n$ matrix inverse is bounded by $O(n^3)$ time.

You may assume $\vec{0}$ is a vertex of the polytope.

You may assume the polytope is non-degenerate (i.e., every vertex has a unique feasible basis).

Hint: The simplex algorithm attempts to find a path from start vertex to optimal vertex by going along edges. We now want to instead find *all* vertices and edges instead of a single path. How can we modify the simplex algorithm so it explores all vertices/edges?

The next pages are ungraded questions. They are questions that would be given in an advanced algorithms class (eg 7000 or 8000 level). If simplex algorithm is a topic that interests you, you can try to solve them for fun.

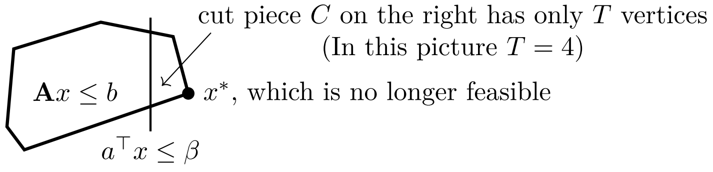
Ungraded Question: An Algorithm for Online Linear Programs

We have a linear program of form $\max c^\top x$ subject to $\mathbf{A}x \le b$ for $\mathbf{A} \in \mathbb{R}^{n\times d}$. Assume we already solved the linear program and have computed the optimal solution x^*.

Next, a new constraint $a^\top x \le \beta$ arrives ($a \in \mathbb{R}^d$, $\beta \in \mathbb{R}$) and we consider the new linear program

$$\max c^\top x \text{ subject to } \mathbf{A}x \le b \text{ and } a^\top x \le \beta. \tag{2}$$

Since we have an extra constraint, the old optimal solution x^* might no longer be valid. Re-solving the problem from scratch could be slow, especially if the new constraint didn't cut off much from the old feasible space. Let us assume that the piece C that was cut off by the new constraint is relatively small and has at most T vertices. (You may even assume that the number of feasible bases for $\mathbf{A}x \le b, a^\top x \ge \beta$ is bounded by T. Note the flipped inequality on $a^\top x \ge \beta$, because this describes the piece that was cut off.)


cut piece C on the right has only T vertices
(In this picture $T = 4$)
$\mathbf{A}x \le b$
x^*, which is no longer feasible
$a^\top x \le \beta$

The next questions will guide you towards an algorithm that can quickly find the new optimal solution, if we already have the old solution x^*.

1. Describe and analyse an algorithm that, given $\mathbf{A}, b, c, x^*, a, \beta$, finds in $O(Tn^3)$ time a feasible vertex for the new linear program (2).
2. Prove that if $a^\top x^* > \beta$, then there is an optimal solution x^{**} for the new linear program (2) that also satisfies $a^\top x^{**} = \beta$.
3. Design and analyse an algorithm that, given $\mathbf{A}, b, c, x^*, a, \beta$, finds in $O(Tn^3)$ the new optimal solution for (2). You may call/use your algorithm from 1, even if you did not solve that question.

You may use the simplex algorithm as blackbox, i.e., your algorithm is allowed to call "SIMPLEX($\mathbf{A}', b', c', x'$)" to solve $\min c^\top x$ subject to $\mathbf{A}'x \le b'$. Here the input x' must be some vertex of the polytope.

Hint: Consider the cut off piece C. It has T corners and x^* is a corner of C.

Remark: This problem is also referred to as "cutting planes" because $a^\top x = \beta$ is a plane that cuts off a part of the polytope. This is a popular framework for solving integer linear programs[2].

[2] https://en.wikipedia.org/wiki/Cutting-plane_method

Ungraded Question: Geometric Interpretation of Dijkstra's Algorithm (General Case)

Let $G = (V, E)$ be a directed graph with positive edge lengths c_e. For simplicity assume the shortest paths are unique (e.g., by adding tiny random noise to each edge weight c_e.) Let s, t be some vertices and we want to compute the distance from s to t. We could do that via Dijkstra's algorithm.

Alternatively, we could solve the following linear program via the Simplex algorithm

$$\begin{aligned} &\max_{x \in \mathbb{R}^V} x_s \\ x_u - x_v &\leq c_{uv} \quad \text{for all } (u,v) \in E \\ x_v &\geq 0 \quad \text{for all } v \in V \setminus \{t\} \\ x_t &\leq 0 \end{aligned}$$

In this exercise, we want to compare Dijkstra's algorithm and the Simplex algorithm.

Consider the following small modification of Dijkstra's algorithm. The changes are highlighted (blue).

Algorithm 2: Dijkstra

```
procedure Dijkstra($G = (V, E), s, t$)
    Queue.insert(s, 0)
    while Queue not empty do
        $v$, d = Queue.pop() // priority queue
        if visited[$v$] then
            continue
        visited[$v$] = true
        distance[$v$] = d
        Construct a vector $x \in \mathbb{R}^V$ with $x_z$ = d − distance[$z$] for all visited $z \in V$, and $x_z = 0$ for unvisited $z \in V$.
        if $v == t$ then return $d$ // We found the distance to $t$, so we can terminate;
        for neighbor $u$ of $v$ do
            Queue.insert(u, d+$c_{vu}$)
```

Let $x^1, x^2, x^3, \ldots \in \mathbb{R}^V$ be the sequence of vectors constructed by Dijkstra's algorithm above (blue lines).

1. What does the first vector x^1 look like?
2. Show that each x^i is a feasible vector for above linear program, and there are $|V|$ constraints that are tight (satisfied with equality).
3. Consider the tight constraints of x^i and the tight constraints of x^{i-1}. How different are those two sets of tight constraints? How many tight constraints do they share?
4. Consider the tight constraints of x^i. Explain why the set of tight constraints is linearly independent.
5. Assume we run the simplex algorithm on the linear program above from starting point 0-vector. Let $y^1, y^2, y^3, \ldots \in \mathbb{R}^V$ be the sequence of corners of the polytope visited by Simplex algorithm. Compare this sequence to $x^1, x^2, x^3, \ldots$. Are they the same or different? Why? (Short 1-2 sentence answer suffices.)

Ungraded Question: Geometric Interpretation of Augmenting Paths

Let $G = (V, E)$ be a bipartite graph and consider the linear program representation of matching:

$$\max \sum_e x_e$$

$$\sum_{u,\text{ neighbor of } v} x_{uv} \leq 1 \quad \text{for all } v \in V \tag{3}$$

$$x_e \geq 0 \quad \text{for all } e \in E \tag{4}$$

1. Let $M \subseteq E$ be a matching (M does not need to be maximum) and let $x \in \mathbb{R}^E$ with $x_e = 1$ if $e \in M$ and $x_e = 0$ otherwise.
   Give a subset of $|E|$ tight constraints (3) and (4) that is linearly independent. (You do not need to prove your set is linearly independent, just listing the set suffices.)

   Remark. *Answer for 2 is also an answer for 1.*

2. Consider a matching M, an augmenting path p, and the matching M' obtained by augmenting M by p. Let $x, x' \in \mathbb{R}^E$ be the vectors corresponding to M and M'.
   Give two subsets B, B' of linear independent constraints (3) and (4). Both B and B' must contain $|E|$ many constraints, and the constraints in B, B' must be tight for x and x' respectively. Further, B' can be obtained from B by replacing only one constraint, i.e., $|B \Delta B'| = 2$. (You only need to list the sets, you do not need to prove the linear independence.)

Remark: 1. Shows x is a corner of the polytope, and 2. shows the corners are connected by an edge of the polytope. The simplex algorithm moves along edges from corner to corner of the feasible polytope. Above exercise shows that running augmenting paths to find the maximum matching does the same thing. The main difference is that the simplex algorithm uses linear algebra and geometry to compute the augmenting paths, whereas algorithms like Ford-Fulkerson use BFS/DFS.

The fastest algorithms for flows and bipartite matching use a combination of both linear algebra and graph perspective[3].

[3] See, e.g., https://arxiv.org/pdf/2203.00671 and https://arxiv.org/pdf/2009.01802.