

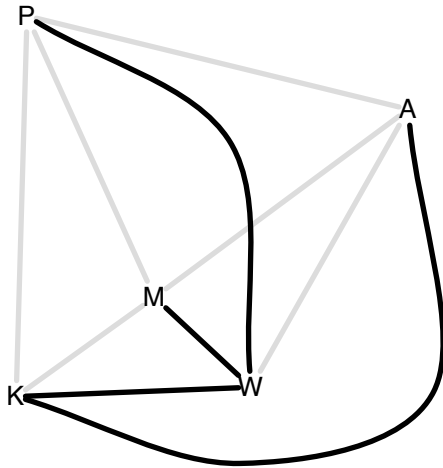
hw5

Coding the Matrix, Summer 2013

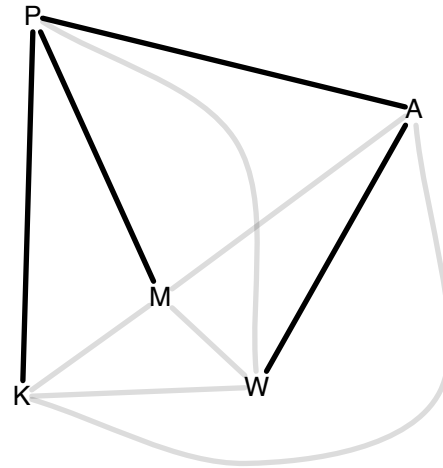
Please fill out the stencil file named “hw5.py”. While we encourage you to complete the Ungraded Problems, they do not require any entry into your stencil file.

Ungraded Problem: In this problem, you will practice using the Exchange Lemma to transform one spanning forest into another.

Consider the graphs below. Use the Exchange Lemma for spanning trees to transform a spanning forest $F_0 = \{(W, K), (W, M), (P, W), (K, A)\}$ on the left into the spanning forest $F_4 = \{(P, K), (P, M), (P, A), (W, A)\}$ on the right. You should draw forests F_0, F_1, F_2, F_3 and F_4 to show each step of your transformation.



(a) Graph with spanning forest F_0 .



(b) Graph with spanning forest F_4 .

For the next two problems, use the Exchange Lemma iteratively to transform a set $S = \{w_0, w_1, w_2\}$ into a set $B = \{v_0, v_1, v_2\}$. In each step, one vector of B is injected, and one vector of S is ejected. Be careful to ensure that the ejection does not change the set of vectors spanned.

You might find the following table useful in keeping track of the iterations.

| | S_i | A | v to inject | w to eject |
|---------|---------------------|---------------------|---------------|--------------|
| $i = 0$ | $\{w_0, w_1, w_2\}$ | \emptyset | | |
| $i = 1$ | | | | |
| $i = 2$ | | | | |
| $i = 3$ | $\{v_0, v_1, v_2\}$ | $\{v_0, v_1, v_2\}$ | - | |

You are to specify the list of vectors comprising S_1 (after one iteration) and S_2 (after two iterations) in the process of transforming from $\{w_0, w_1, w_2\}$ to $\{v_0, v_1, v_2\}$.

Problem 1: Vectors over \mathbb{R} :

$$\mathbf{w}_0 = [1, 0, 0]$$

$$\mathbf{w}_1 = [0, 1, 0]$$

$$\mathbf{w}_2 = [0, 0, 1]$$

$$\mathbf{v}_0 = [1, 2, 3]$$

$$\mathbf{v}_1 = [1, 3, 3]$$

$$\mathbf{v}_2 = [0, 3, 3]$$

Problem 2: Vectors over $GF(2)$:

$$\mathbf{w}_0 = [0, one, 0]$$

$$\mathbf{w}_1 = [0, 0, one]$$

$$\mathbf{w}_2 = [one, one, one]$$

$$\mathbf{v}_0 = [one, 0, one]$$

$$\mathbf{v}_1 = [one, 0, 0]$$

$$\mathbf{v}_2 = [one, one, 0]$$

Problem 3: In this problem, you will write a procedure to achieve the following goal:

- *input*: a list S of vectors, and a list B of linearly independent vectors such that $\text{Span } S = \text{Span } B$
- *output*: a list T of vectors that includes B and possibly some vectors of S such that
 - $|T| = |S|$, and
 - $\text{Span } T = \text{Span } S$

This is not useful in its own sake, and indeed there is a trivial implementation in which T is defined to consist of the vectors in B together with enough vectors of S to make $|T| = |S|$. The point of writing this procedure is to illustrate your understanding of the proof of the Morphing Lemma. The procedure should therefore mimic that proof: T should be obtained step by step from S by, in each iteration, injecting a vector of B and ejecting a vector of $S - B$ using the Exchange Lemma. The procedure must return the list of pairs (injected vector, ejected vector) used in morphing S into T .

The procedure is to be called `morph(S, B)`. The spec is as follows:

- *input*: a list S of distinct vectors, and a list B of linearly independent vectors such that $\text{Span } S = \text{Span } B$
- *output*: a k -element list $[(z_1, w_1), (z_2, w_2), \dots, (z_k, w_k)]$ of pairs of vectors such that, for $i = 1, 2, \dots, k$,

$$\text{Span } S = \text{Span } S \cup \{z_1, z_2, \dots, z_i\} - \{w_1, w_2, \dots, w_k\}$$

where $k = |B|$.

This procedure uses a loop. You can use the procedure `exchange(S, A, z)` or the procedure `vec2rep(veclist, u)`, both from Homework 4, or the `solver` module.

Here is an illustration of how the procedure is used.

```
>>> S = [list2vec(v) for v in [[2,4,0],[1,0,3],[0,4,4],[1,1,1]]]
>>> B = [list2vec(v) for v in [[1,0,0],[0,1,0],[0,0,1]]]
>>> for (z,w) in morph(S, B):
...   print("injecting ", z)
...   print("ejecting ", w)
...   print()
... 
```

injecting

0 1 2

1 0 0

ejecting

0 1 2

2 4 0

injecting

0 1 2

0 1 0

ejecting

0 1 2

1 0 3

injecting

0 1 2

0 0 1

ejecting

0 1 2

0 4 4

Test your procedure with the above example. Your results need not exactly match the results above.

Problem 4: For each of the following matrices, (a) give a basis for the row space (b) give a basis for the column space, and (c) verify that the row rank equals the column rank. Justify your answers.

1. $\begin{bmatrix} 1 & 2 & 0 \\ 0 & 2 & 1 \end{bmatrix}$

2. $\begin{bmatrix} 1 & 4 & 0 & 0 \\ 0 & 2 & 2 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$

3. $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

4. $\begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 3 & 4 \end{bmatrix}$

Problem 5: In this problem you will again write an independence-testing procedure. Write and test a procedure `my_is_independent(L)` with the following spec:

- *input*: a list L of vectors
- *output*: True if the vectors form a linearly independent list.

Vectors are represented as instances of `Vec`. We have provided a module `independence` that provides a procedure `rank(L)`. You should use this procedure to write `my_is_independent(L)`. No loop or comprehension is needed. This is a very simple procedure.

Problem 6: Write and test a procedure `subset_basis(T)` with the following spec:

- *input*: a list T of vectors
- *output*: a list S consisting of vectors of T such that S is a basis for the span of T .

Your procedure should be based on either a version of the Grow algorithm or a version of the Shrink algorithm. Think about each one to see which is easier for you. You will need a loop or comprehension for this procedure. You can use as a subroutine any one of the following:

- the procedure `is_superfluous(L, b)` from HW4, or
- the procedure `is_independent(L)` from Problem ?? or from the module `independence` we provide, or
- the procedure `solve(A,b)` from the `solver` module.

Problem 7: Write and test a procedure `my_rank(L)` with the following spec:

- *input*: a list L of Vectors
- *output*: The rank of L

You can use the procedure `subset_basis(T)` from Problem 6, in which case no loop is needed. Alternatively, you can use the procedure `is_independent(L)` from the module `independence` we provide; in this case, the procedure requires a loop.

Problem 8: Each of the following subproblems specifies two subspaces \mathcal{U} and \mathcal{V} of a vector space. For each subproblem, check whether $\mathcal{U} \cap \mathcal{V} = \{\mathbf{0}\}$.

1. Subspaces of $GF(2)^4$: let $\mathcal{U} = \text{Span} \{1010, 0010\}$ and let $\mathcal{V} = \text{Span} \{0101, 0001\}$.
2. Subspaces of \mathbb{R}^3 : let $\mathcal{U} = \text{Span} \{[1, 2, 3], [1, 2, 0]\}$ and let $\mathcal{V} = \text{Span} \{[2, 1, 3], [2, 1, 3]\}$.
3. Subspaces of \mathbb{R}^4 : let $\mathcal{U} = \text{Span} \{[2, 0, 8, 0], [1, 1, 4, 0]\}$ and let $\mathcal{V} = \text{Span} \{[2, 1, 1, 1], [0, 1, 1, 1]\}$

Problem 9: Write and test a procedure `direct_sum_decompose(U_basis, V_basis, w)` with the following spec:

- *input*: A list U_basis containing a basis for a vector space \mathcal{U} , a list V_basis containing a basis for a vector space \mathcal{V} , and a vector w that belongs to the direct sum $\mathcal{U} \oplus \mathcal{V}$
- *output*: a pair (u, v) such that $w = u + v$ and u belongs to \mathcal{U} and v belongs to \mathcal{V} .

All vectors are represented as instances of `Vec`. Your procedure should use the fact that a basis of \mathcal{U} joined with a basis of \mathcal{V} is a basis for $\mathcal{U} \oplus \mathcal{V}$. It should use the `solver` module or the procedure `vec2rep(veclist, u)` from `The Basis`.

Problem 10: Write and test a procedure `is_invertible(M)` with the following spec:

- *input:* an instance `M` of `Mat`
- *output:* *True* if `M` is an invertible matrix, *False* otherwise.

Your procedure should not use any loops or comprehensions. It can use procedures from the `matutil` module and from the `independence` module.

Problem 11: Write a procedure `find_matrix_inverse(A)` with the following spec:

- *input:* an invertible matrix A over $GF(2)$ (represented as a `Mat`)
- *output:* the inverse of A (also represented as a `Mat`)

Note that the input and output matrices are over $GF(2)$.

Your procedure should use as a subroutine the `solve` procedure of the `solver` module. Since we are using $GF(2)$, you need not worry about rounding errors. Your procedure should be based on the following result:

Suppose A and B are square matrices such that AB is the identity matrix. Then A and B are inverses of each other.

In particular, your procedure should try to find a square matrix B such that AB is an identity matrix:

$$\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} \begin{bmatrix} & & \\ & B & \\ & & \end{bmatrix} = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}$$

To do this, consider B and the identity matrix as consisting of columns.

$$\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} \begin{bmatrix} \left| \begin{array}{c} b_1 \end{array} \right| & \cdots & \left| \begin{array}{c} b_n \end{array} \right| \end{bmatrix} = \begin{bmatrix} \left| \begin{array}{c} 1 \\ \vdots \\ 1 \end{array} \right| & \cdots & \left| \begin{array}{c} 1 \\ \vdots \\ 1 \end{array} \right| \end{bmatrix}$$

Using the matrix-vector definition of matrix-matrix multiplication, you can interpret this matrix-matrix equation as a collection of n matrix-vector equations: one for b_1 , ..., one for b_n . By solving these equations, you can thus obtain the columns of B .

Remember: If A is an $R \times C$ matrix then AB must be an $R \times R$ matrix so the inverse B must be a $C \times R$ matrix.

Problem 12: You will write a procedure for finding the inverse of an upper-triangular matrix.

`find_triangular_matrix_inverse(A)`

- *input*: an instance M of `Mat` representing an upper triangular matrix with nonzero diagonal elements. You can assume that the row-label set and column-label set are of the form $\{0, 1, 2 \dots, n - 1\}$.
- *output*: a `Mat` representing the inverse of M

This procedure should use `triangular_solve` which is defined in the module `triangular`. It can also use the procedures in `matutil`, but that's all.