CPSC 320 2017W1: Assignment 5

November 22, 2017

Please submit this assignment via GradeScope at `https://gradescope.com`. Be sure to identify everyone in your group if you're making a group submission (which we encourage!).

Submit by the deadline **Friday 1 Dec at 10PM**. For credit, your group must make a **single** submission via one group member's account, marking all other group members in that submission **using GradeScope's interface**. Your group's submission **must**:

- Be on time.
- Consist of a single, clearly legible file uploadable to GradeScope with clearly indicated solutions to the problems. (PDFs produced via LaTeX, Word, Google Docs, or other editing software work well. Scanned documents will likely work well. **High-quality** photographs are OK if we agree they're legible.)
- Include prominent numbering that corresponds to the numbering used in this assignment handout (not the individual quizzes). Put these **in order** starting each problem on a new page, ideally. If not, **very clearly** and prominently indicate which problem is answered where!
- Include at the start of the document the **ugrad.cs.ubc.ca e-mail addresses** of each member of your team. (No names are necessary.)
- Include at the start of the document the statement: "All group members have read and followed the guidelines for academic conduct in CPSC 320. As part of those rules, when collaborating with anyone outside my group, (1) I and my collaborators took no record but names (and GradeScope information) away, and (2) after a suitable break, my group created the assignment I am submitting without help from anyone other than the course staff." (Go read those guidelines!)
- Include at the start of the document your outside-group collaborators' ugrad.cs.ubc.ca e-mail addresses. (Be sure to get those when you collaborate!)

(Do **not** include your *name* (or collaborators' names) on the submitted PDF/images of your assignment.)

1 Carvin' the Seams

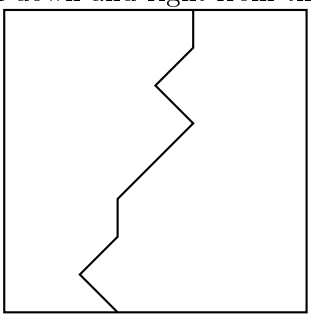
You can resize an image by scaling or cropping it, but what if the pieces of the image that you want are not all in one rectangular area, and you don't want to make those parts of the image smaller by scaling?[1]

In that case, you might instead choose to eliminate one pixel from each row (to make the image one pixel narrower) or one pixel from each column (to make the image shorter) while somehow optimizing for the "best" pixels to remove. In this problem, we focus on removing one pixel from each row.

We'll assume an image is an n row by m column array of pixels $A[1 \ldots n][1 \ldots m]$, where each pixel is an "energy" rather than a color. Energies are non-negative numbers representing the importance of the pixel.

A legal seam must include one pixel from every row. Each pair of seam pixels in neighbouring rows must be either in the same column or one column apart (i.e., on a diagonal). The cost of a seam is the total energy of all the pixels in the seam. The best seam is the one with lowest cost.

So, a seam of pixels to remove often looks a little like a "lightning bolt" moving down, down-and-left, and down-and-right from the top to the bottom of the image, such as this:



[1] This method was developed by Avidan and Shamir.

1. Circle two non-overlapping seams in this diagram that have different costs. Indicate their costs and which seam is better. One of the two should be the optimal seam in the diagram.

```
1 8 7 5 6 2 4
9 5 1 2 8 8 7
6 6 2 1 9 5 4
```

2. Complete the partial recurrence below (which has only the recursive case) for the cost of the best partial seam that has pixels only from the top row 1 down to row i and ends at the pixel in row i and column j. Your recurrence should be in terms of the seams ending at pixels in the row above, row $i-1$. Assume $1 < i \leq n$ and $1 \leq j \leq m$.

```
C(i, j) = the _____ of these options:

        C(____, ____) if j > _____ (else infinity),

        C(____, ____), and

        C(____, ____) if j < _____ (else infinity)
```

3. Give the cost for a partial seam that only has a pixel in the very first row, $i = 1$. (This is our base case.)

```
C(1, j) =
```

4. Assuming that $C(i, j)$ has already been implemented correctly for $1 \leq i \leq n$ and $1 \leq j \leq m$, complete the following code that finds the cost of the optimal seam to remove from an $n \times m$ image. You should **certainly** refer to C within `OptCost`. You may assume that C just "magically" has access to `A`, since there's no good way to pass it in! ☺

```
OptCost(A, n, m):
```

5. Asymptotically (in terms of n and/or m), how many subproblems must an implementation of `OptCost` using a memoized implementation of C solve? Fill in the circle next to the **best** answer.

- ◯ $O(1)$
- ◯ $O(\log_3 n)$
- ◯ $O(\log_3 m)$
- ◯ $O(n)$
- ◯ $O(m)$
- ◯ $O(n^2)$
- ◯ $O(m^2)$
- ◯ $O(nm)$
- ◯ $O(3^n)$
- ◯ $O(3^m)$

6. Asymptotically (in terms of n and/or m), how long does it take to solve each subproblem $C(i, j)$, not counting the runtime of recursive calls that subproblem makes? Fill in the circle next to the **best** answer.

- ◯ $O(1)$
- ◯ $O(\log_3 i)$
- ◯ $O(\log_3 j)$
- ◯ $O(i)$
- ◯ $O(j)$
- ◯ $O(im)$
- ◯ $O(jn)$
- ◯ $O(ij)$
- ◯ $O(3^i)$
- ◯ $O(3^j)$

1.1 Quiz Solution

1. There are **many** answers for one of the seams. All of them must have two sets of seams that contain entirely distinct pixels, each of which has one pixel per row and never shifts left or right as it goes down a row by more than one space. However, all correct solutions **must** include the seam of total cost 7. Here's one pair of seams:

$$\begin{array}{cccccccc} 1 & 8 & 7 & \textcircled{5} & 6 & 2 & \textcircled{4} \\ 9 & 5 & \textcircled{1} & 2 & 8 & 8 & \textcircled{7} \\ 6 & 6 & 2 & \textcircled{1} & 9 & 5 & \textcircled{4} \end{array}$$

   The left-hand seam is the lowest-cost in this image, with cost $5+1+1=7$ (and so is better than the right-hand one). The right-hand seam has cost $4+7+4=15$.

2. Here is our (partial) recurrence:

```
C(i, j) = the _min_ of these options:

        C(_i-1_, _j-1_) if j > _1_ (else infinity),

        C(_i-1_, _j_), and

        C(_i-1_, _j+1_) if j < _m_ (else infinity)
```

3. Here is our base case:

```
C(1, j) = A[1][j]
```

4. The key insight is that the best seam could end in any column in the last row:

```
OptCost(A, n, m):
    best = infinity

    for j = 1 to m:
        if C(n, j) < best:   // C is memoized and magically
            best = C(n, j)   // has access to A

    return best
```

5. How many subproblems must `OptCost` solve?

   ○ $O(1)$
   ○ $O(\log_3 n)$
   ○ $O(\log_3 m)$
   ○ $O(n)$
   ○ $O(m)$
   ○ $O(n^2)$
   ○ $O(m^2)$
   ● $O(nm)$
   ○ $O(3^n)$
   ○ $O(3^m)$

   Indeed, we actually will fill in **every** entry in the $n \times m$ memoizing table for any image we call `OptCost` on.

6. How long does it take to solve each subproblem $C(i, j)$?

   - (●) $O(1)$
   - ( ) $O(\log_3 i)$
   - ( ) $O(\log_3 j)$
   - ( ) $O(i)$
   - ( ) $O(j)$
   - ( ) $O(im)$
   - ( ) $O(jn)$
   - ( ) $O(ij)$
   - ( ) $O(3^i)$
   - ( ) $O(3^j)$

   We take the minimum of three numbers with some simple conditionals and arithmetic operations to compute those numbers, plus three recursive calls to C.

1.2 Assignment

1. Give a pseudocode algorithm that finds the cost of the best seam in an energy array using dynamic programming (iterative memoization). Your algorithm should look like the following, taking a 2-dimensional array of n rows and m columns that uses 1-based indexing and produce the cost of the best seam through the entire array from top to bottom.

   ```
   OptCost(A, n, m):
   ```

2. Give a pseudocode algorithm that takes the memoizing/dynamic programming table created as part of your `OptCost` function above and produces the column numbers of the pixels in the best seam. (So, if the best seam has a pixel at column 3 in the first row and column 4 in the second, then your solution should give the list $[3, 4]$.)

3. Clearly describe how you could store a small amount of additional information (at most two bits) in each entry of the memoizing table in your algorithm in part 1 in order to make the algorithm in part 2 simpler to write. Your solution should be similar to how a shortest path algorithm can store a predecessor node at each node's entry and use them to reconstruct the path. (You only need to clearly describe the strategy, not rewrite your code.)

BONUS: For one **bonus point**, look up seam carving and design an implementation. Maybe extend it to apply to video! To earn bonus points, you must post your solution (including code) to Piazza privately with the title "Seam Carving Bonus" and a clear discussion of how and why you designed it the way you did. Then, submit the URL of your post in GradeScope.

2 Shreddin' the Ink-Lined Plane

You are with a document recovery company. Given scanned fragments of a shredded document, your job is to reassemble the fragments into the original from which they were shredded. In particular, you're working on the "shredda" (output) from strip-cut shredders that separate a piece of paper into long, side-by-side strips.

For each ordered pair of strips a and b, you have a **non-positive** (i.e., zero or negative) score indicating how likely a is to be the immediate left neighbour of b in the reconstructed document.[2] If the score is 0, a is definitely just left of b. The more negative the score, the less likely a is to be just left of b. You want to find an arrangement of all the strips that maximizes the sum of the scores of each pair of neighboring strips.

For example, if you had three strips a, b, and c, you might have the following scores:

	a	b	c
a		-2	-3
b	-10		-9
c	-4	-7	

The label along the left column indicates the left-hand strip, while the label along the top row indicates the right-hand strip. So, a left of b has a score of -2.

Ordering the strips a, b, c would have a total score of $-2 + -9 = -11$, but the optimal solution is c, a, b with total score $-4 + -2 = -6$.

1. Complete the following brute force recursive solution to this problem. It begins with a helper function `GetScore` that you can then use in `BestDeShredScore` on the next page.

```
// Given the score matrix, produces the score of placing strip i left of j
// i == 0 represents j being the leftmost strip.
GetScore(score, i, j):

    if i == 0:

        return ___

    else:

        return score[i, j]
```

[2] These numbers actually represent "log likelihoods", but that's not critical to us. The problem is really about DNA sequencing, but that's also not critical to us!

```
// score[i,j] (for i and j in the range 1 up to n but i not equal to j) is the
// non-positive score for strip i being the immediate left neighbour of strip j.
BestDeShredScore(score, n):
    // If used[i] is true, then shred i has already been placed.
    // Else, it is still available to be placed. left is the index
    // of the most recently placed (rightmost) strip, or 0 if none
    // have yet been placed. For left > 0, used[left] must be true.
    Helper(used, left):
        all_used = true
        best_score = -infinity

        // Find the best next shred over all unused i.
        for i = 1 to n:
            if used[i] is false:
                all_used = false

                // assume constant time:
                new_used = copy of used but with element i set to true

                best_score = _____(Helper(new_used, i) + ______________________________,

                                   best_score)

        if all_used:

            return ______________

        else:

            return ______________
```

2. Select the best (i.e., tightest) **lower-bound** from among the options below on the runtime of this algorithm in terms of the number of strips n. (Note: the bound you choose may not be *tight*, but it should be the tightest available.)

   ◯ $\Omega(1)$
   ◯ $\Omega(n)$
   ◯ $\Omega(n^2)$
   ◯ $\Omega(2^n)$
   ◯ $\Omega(n!)$
   ◯ $\Omega(n^n)$

3. We might consider memoizing the function `Helper`. Would it be useful to memoize it? Fill in the circle next to the **best** answer.

   ◯ Yes. Memoization would give us (at least) exponential speedup for (at most) polynomial memory.
   ◯ Yes. Memoization would give us (at least) exponential speedup for (at least) exponential memory.
   ◯ No. A given call to `Helper` does not repeat subproblems; so, memozition is not useful.
   ◯ No. `Helper` takes arguments that cannot be memoized; so, memoization is not useful.

ON THE NEXT PAGE, we'd like to reason about whether this problem is NP-complete, but the NP-complete problems are a set of **decision** problems, problems whose answers are `Yes` or `No`. So, we must convert this problem to a decision problem.

4. Which of these is the most promising way to change our optimal shredda reassembly problem into a decision problem, in the sense that it captures the core features of the problem? Fill in the circle next to the **best** answer.

   ◯ Add non-positive parameter k; ask if any arrangement of the full set of strips produces a score $\geq k$.
   ◯ Add non-positive parameter k; ask if any arrangement of the full set of strips produces a score $\leq k$.
   ◯ Add non-negative parameter k; ask if any arrangement using $\geq k$ of the strips produces a score of zero.
   ◯ Add non-negative parameter k; ask if any arrangement using $\leq k$ of the strips produces a score of zero.

5. Considering the same set of ways to change the optimal shredda reassembly problem into a decision problem, which is the **least** promising in the sense that it always has the same solution, independent of the instance? Fill in the circle next to the **least promising** answer.

   ◯ Add non-positive parameter k; ask if any arrangement of the full set of strips produces a score $\geq k$.
   ◯ Add non-positive parameter k; ask if any arrangement of the full set of strips produces a score $\leq k$.
   ◯ Add non-negative parameter k; ask if any arrangement using $\geq k$ of the strips produces a score of zero.
   ◯ Add non-negative parameter k; ask if any arrangement using $\leq k$ of the strips produces a score of zero.

6. For whatever **most promising** decision variant you choose, we'll want to prove that it is in NP, which requires defining a certificate—a short piece of data that can be used to quickly establish that the answer to an instance is `Yes`. Fill in the blanks to indicate what a good certificate would be.

   My certificate will be a [ ] of [ ] that indicates [ ].

7. Briefly but clearly complete the polynomial time algorithm below that takes an instance of your decision problem and a certificate (as you described above) and confirms (or denies) that the answer to the instance is `Yes`.

```
// Given an instance i of the shredda problem and a certificate c for i,
// produce TRUE if c can be used to prove that i's answer is Yes and FALSE otherwise.
Certify(i, c):
    // Note: i has the form of ____________________________________________
    //
    //       c has the form of ____________________________________________




    // Validate the STRUCTURE of c:









    // Ensure that c fulfills the constraints on the instance
```

2.1 Quiz Solution

1. Here is a completed algorithm:

```
GetScore(score, i, j):
    if i == 0:
        return _0_    // used for the first strip placed
    else:
        return score[i, j]

// score[i,j] (for i and j in the range 1 up to n but i not equal to j) is the
// non-positive score for strip i being the immediate left neighbour of strip j.
BestDeShredScore(score, n):
    // If used[i] is true, then shred i has already been placed.
    // Else, it is still available to be placed. left is the index
    // of the most recently placed (rightmost) strip, or 0 if none
    // have yet been placed. For left > 0, used[left] must be true.
    Helper(used, left):
        all_used = true
        best_score = -infinity

        // Find the best next shred over all unused i.
        for i = 1 to n:
            if used[i] is false:
                all_used = false

                // assume constant time:
                new_used = copy of used but with element i set to true

                // The best score is the LARGEST, and if i is the correct strip
                // to place next, then the score is the score of the remaining
                // strips starting with i on the left plus the score of laying i
                // down next to left.
                best_score = _max_(Helper(new_used, i) + _GetScore(score, left, i)_,
                                   best_score)

        if all_used:
            return _0_           // the score of no strips is 0
        else:
            return _best_score_  // otherwise, the best score we found!
```

2. Here is the tightest available lower-bound:

- ○ $\Omega(1)$
- ○ $\Omega(n)$
- ○ $\Omega(n^2)$
- ○ $\Omega(2^n)$
- ● $\Omega(n!)$
- ○ $\Omega(n^n)$

You'll develop a recurrence for the runtime, draw a recurrence tree, and find a better bound as part of the assignment.

3. Would it be useful to memoize `Helper`?

- ○ Yes. Memoization would give us (at least) exponential speedup for (at most) polynomial memory.
- ● Yes. Memoization would give us (at least) exponential speedup for (at least) exponential memory.
- ○ No. A given call to `Helper` does not repeat subproblems; so, memozition is not useful.
- ○ No. `Helper` takes arguments that cannot be memoized; so, memoization is not useful.

We definitely solve many repeated subproblems (as when we place strip a then b then c vs. b then a then c). However, the set of problems we solve is defined by the subset of the strips that can be used at any given time along with the second parameter. There are an exponential number of subsets of the set of strips, not even considering the second parameter! Overall, this does give us exponential speedup (in fact, super-exponential), but at the expense of exponential memory use.

As for the last answer: We can certainly use a tuple of boolean values and a number together as a key in a hash table (or even as indexes in a multi-dimensional array).

4. Which of these is the most promising decision variant of the problem, in the sense that it captures the core features of the problem?

- ● Add non-positive parameter k; ask if any arrangement of the full set of strips produces a score $\geq k$.
- ○ Add non-positive parameter k; ask if any arrangement of the full set of strips produces a score $\leq k$.
- ○ Add non-negative parameter k; ask if any arrangement using $\geq k$ of the strips produces a score of zero.
- ○ Add non-negative parameter k; ask if any arrangement using $\leq k$ of the strips produces a score of zero.

This captures the idea of reassembling all the strips to maximize score. The answer that uses $\geq k$ strips to achieve a score of zero at least captures a special case of the problem. (If we *can* achieve a score of zero, we do want to.) We'll discuss the last two answers next.

5. Which is the **least** promising decision variant of the problem?

- ○ Add non-positive parameter k; ask if any arrangement of the full set of strips produces a score $\geq k$.
- ○ Add non-positive parameter k; ask if any arrangement of the full set of strips produces a score $\leq k$.
- ○ Add non-negative parameter k; ask if any arrangement using $\geq k$ of the strips produces a score of zero.
- ● Add non-negative parameter k; ask if any arrangement using $\leq k$ of the strips produces a score of zero.

We already said that first and third answers are the most promising. Which of the second and fourth is least promising? They're both pretty bad, but at least the second problem captures some element of the problem. We need to *find* some arrangement that produces a low score. Maybe all arrangements have higher scores than the threshold.

Compare that to the fourth answer. No matter what instance you give it, if you use 0 strips, that will be $\leq k$ for a non-negative k and will produce a score of zero. So, the answer to *every* instance of this decision problem is `Yes`.

6. Fill in the blanks to indicate a good certificate for the problem:

My certificate will be a [list] of [strips] that indicates [their arrangement/order].

7. Certifying algorithm:

```
// Given an instance i of the shredda problem and a certificate c for i,
// produce TRUE if c can be used to prove that i's answer is Yes and FALSE otherwise.
Certify(i, c):
    // Note: i has the form of _a_tuple_of_a_nxn_score_array_and_k_
    //       c has the form of _list_of_numbers_   // which should be a permutation of 1..n
    let (score,k) = i
```

```
let n = length(score) // size of one dimension of score

// Validate the STRUCTURE of c:
if length(c) != n: return FALSE
let sorted_c = efficient_sort(c)
if sorted_c != [1..n]: return FALSE

// Ensure that c fulfills the constraints on the instance
total_score = 0
for j = 1 to n-1:
    total_score += score[c[j],c[j+1]]

return total_score >= k:
```

2.2 Assignment

1. Give a recurrence $T(i, n)$ for the runtime of the brute force `BestDeShredScore` algorithm (without memoization). Let n be the total number of strips in the original instance and i be the number of strips remaining under consideration (i.e., the number of false entries in `used`). Assume $0 \leq i \leq n$.

2. Draw enough of a recursion tree for $T(n, n)$ to develop a good asymptotic bound on the runtime of the brute force `BestDeShredScore` algorithm in terms of n. Annotate all key elements in your tree, including the work in each node (not counting recursive calls) and the total work per level. (We ask for the bound itself next; the solution to this problem is only the tree with the information needed to develop the bound.)

3. Give a good Θ bound on the runtime of `BestDeShredScore` in terms of n based on your recursion tree. You **need not** prove your bound correct (but it should **be** correct!).

4. Give a tight big-O bound on the number of distinct subproblems solved by `Helper` in a call to `BestDeShredScore` on a problem with n strips.

5. Give a tight big-O bound on the runtime of a memoized version of `BestDeShredScore`. **ASSUME** that you can look up a particular subproblem in the memoizing table in constant time. (It's not at all obvious that this assumption is reasonable, but we'll run with it.)

6. We've already proven that the decision variant of the shredda reassembly problem (hereafter, SRP) is in NP by providing a certificate that can be verified in time polynomial in the length of the instance with which it is associated.

   Now, prove that SRP is NP-hard by reducing from the graph-style Travelling Salesperson Problem (TSP) to SRP. This variant of TSP takes as input a complete (with no self-loops), directed, weighted graph $G = (V, E, w)$ (where w is a function $E \to \mathbb{N}$ mapping edges to their non-negative weights), a starting vertex $s \in V$, and a threshold $k \geq 0$ and determines whether a simple cycle of length $|V|$ exists starting at s with total weight along its edges $\leq k$. You may assume $|V| > 1$. (If graphs with $|V| \leq 1$ caused trouble, we could simply solve them as special cases in constant time.)

   (Note that together knowing that SRP $\in$ NP and SRP $\in$ NP-hard proves SRP $\in$ NP-complete.)

   Specifically, give a clear reduction from TSP to SRP. Then **sketch the key points** in a proof that your reduction is correct. (I.e., the answer to the SRP instance your reduction produces from a given TSP instance is `Yes` if and only if the answer to that TSP instance is `Yes`.)

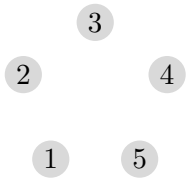
3 Between Rocks and Hard Places

A graph $G = (V, E)$, is said to be k-colorable if its vertices can be labeled by k colors so that no two adjacent vertices are labeled with the same color. The k-COLOR decision problem takes as input a graph G, and returns YES if G is k-colorable, and NO otherwise.

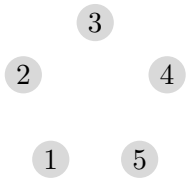
In this quiz we explore the 2, 3, and 12-COLOR problems.

2-COLOR:

1. Add 6 edges to the vertices below to form a graph that is 2-colorable.



2. Add 6 edges to the vertices below to form a graph that is NOT 2-colorable.



3. A simple change to which of the following algorithms is the basis for an efficient solution to the 2-COLOR problem?
   - ◯ Breadth First Search
   - ◯ Gale-Shapley Stable Matching
   - ◯ Dijkstra's Shortest Path
   - ◯ Deterministic Select

4. Given a graph $G = (V, E)$ with $|V| = n, |E| = m$, the running time of an efficient algorithm to find a 2-coloring of G is:
   - ◯ $\Theta(\log(n + m))$
   - ◯ $\Theta(m \log n)$
   - ◯ $\Theta(m)$
   - ◯ $\Theta(n + m)$
   - ◯ $\Theta(n^2)$
   - ◯ $\Theta(mn)$

12-COLOR:

Suppose a reliable authority figure (an algorithm oracle) tells you that there is no polynomial time algorithm to solve the 3-COLOR problem. In this part of the problem we will explore the relationships between 2-COLOR, 3-COLOR, and previously undiscussed 12-COLOR. In the left column of the table below, we have listed a sequence of polynomial time reductions from an instance of problem A to an instance of problem B, using symbols: $A \leq_P B$. In the right column, select the statement that most accurately reflects the significant, new information established by the reduction. Don't forget to use 1) the assumption that 3-COLOR has no polynomial time algorithm, and 2) the insight into 2-COLOR you gained in the previous part of the quiz.

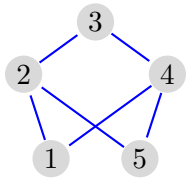
$A \leq_P B$	Answer
12-$COLOR \leq_P$ 3-$COLOR$	◯ A has a polynomial time algorithm. ◯ B has a polynomial time algorithm. ◯ A has no polynomial time algorithm. ◯ B has no polynomial time algorithm. ◯ None of these statements is significant, new information established by the reduction.
12-$COLOR \leq_P$ 2-$COLOR$	◯ A has a polynomial time algorithm. ◯ B has a polynomial time algorithm. ◯ A has no polynomial time algorithm. ◯ B has no polynomial time algorithm. ◯ None of these statements is significant, new information established by the reduction.
3-$COLOR \leq_P$ 12-$COLOR$	◯ A has a polynomial time algorithm. ◯ B has a polynomial time algorithm. ◯ A has no polynomial time algorithm. ◯ B has no polynomial time algorithm. ◯ None of these statements is significant, new information established by the reduction.
2-$COLOR \leq_P$ 12-$COLOR$	◯ A has a polynomial time algorithm. ◯ B has a polynomial time algorithm. ◯ A has no polynomial time algorithm. ◯ B has no polynomial time algorithm. ◯ None of these statements is significant, new information established by the reduction.

3.1 Quiz Solution

2-COLOR:

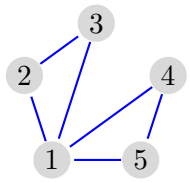
1. Add 6 edges to the vertices below to form a graph that is 2-colorable.

   Here is one among many possible solutions:



2. Add 6 edges to the vertices below to form a graph that is NOT 2-colorable.

   Again, here is one among many possible solutions:



3. A simple change to which of the following algorithms is the basis for an efficient solution to the 2-COLOR problem?

   - (●) Breadth First Search
   - ( ) Gale-Shapley Stable Matching
   - ( ) Dijkstra's Shortest Path
   - ( ) Deterministic Select

   SOLUTION NOTES: Color alternating levels the same colour. Check that no node has an edge to another node in its level. (We are guaranteed that no node has an edge to another node more than one level away. Review the K&T section on BFS if you don't recall why!)

4. Given a graph $G = (V, E)$ with $|V| = n, |E| = m$, the running time of an efficient algorithm to find a 2-coloring of G is:

   - ( ) $\Theta(\log(n+m))$
   - ( ) $\Theta(m \log n)$
   - ( ) $\Theta(m)$
   - (●) $\Theta(n+m)$
   - ( ) $\Theta(n^2)$
   - ( ) $\Theta(mn)$

12-COLOR:

Assuming there is no polynomial time algorithm to solve the 3-COLOR problem:

$A \leq_P B$	Answer
12-$COLOR \leq_P$ 3-$COLOR$	○ A has a polynomial time algorithm. ○ B has a polynomial time algorithm. ○ A has no polynomial time algorithm. ○ B has no polynomial time algorithm. ● None of these statements is significant, new information established by the reduction.
12-$COLOR \leq_P$ 2-$COLOR$	● A has a polynomial time algorithm. ○ B has a polynomial time algorithm. ○ A has no polynomial time algorithm. ○ B has no polynomial time algorithm. ○ None of these statements is significant, new information established by the reduction.
3-$COLOR \leq_P$ 12-$COLOR$	○ A has a polynomial time algorithm. ○ B has a polynomial time algorithm. ○ A has no polynomial time algorithm. ● B has no polynomial time algorithm. ○ None of these statements is significant, new information established by the reduction.
2-$COLOR \leq_P$ 12-$COLOR$	○ A has a polynomial time algorithm. ○ B has a polynomial time algorithm. ○ A has no polynomial time algorithm. ○ B has no polynomial time algorithm. ● None of these statements is significant, new information established by the reduction.

3.2 Assignment

Prove that 12-COLOR is NP-Complete:

1. Write pseudocode for an algorithm that takes as input a graph, $G = (V, E)$, and a vertex labelling L, and returns TRUE if L is a 12 coloring of G, and FALSE otherwise. (We are leaving it to you to describe the form of L. There are a few reasonable options.)
2. What is the running time of your algorithm in the previous part in terms of $|V|$ and $|E|$?
3. **Is 12-COLOR $\in NP$?**
4. Reduce 3-COLOR to 12-COLOR. That is, describe an algorithm that takes as input an instance x of 3-COLOR, and returns an instance y of 12-COLOR so that x has a 3 coloring if and only if y has a 12 coloring.
5. What is the running time of your reduction in terms of $|V|$ and $|E|$?
6. How much space does your reduction use in terms of $|V|$ and $|E|$?
7. **Is this a polynomial reduction?**
8. Prove that your reduction from 3-COLOR to 12-COLOR is correct.
9. **Make a closing statement about 12-COLOR.**

4 Falling Apart

The **kPartition** decision problem is defined as follows: Given a set S of $k \cdot n$ positive integers, return YES if the elements of S can be split into n subsets of k elements each in such a way that all of the subset sums are equal, and NO otherwise.

In this quiz we will explore the relationships between 2, 3, and 12Partition. By the term *set-sum* we mean the sum of the elements of a set.

Group Pre-Quiz:

1. Partition the following set of 12 integers into 6 sets of size 2, all 6 of which have the same set-sum.

   While you do this simple task, try to envision an algorithm by which it can be done.

$$\{78, 187, 178, 22, 54, 13, 141, 72, 146, 122, 59, 128\}$$

2. Give an instance of 3Partition with $n = 4$, for which 3Partition returns YES.

3. Provide evidence that your instance from the previous part results in YES by listing the subsets. (This is sometimes referred to as a *certificate* of the decision problem, or sometimes it's referred to as *solution* to the underlying problem.)

Individual Quiz:

1. Give a brief step-by-step description of an algorithm to solve the 2Partition problem. Your algorithm must run in time $O(n^3)$, and may very well do better (ours does).

2. What is the asymptotic running time of your algorithm?

Suppose a reliable authority figure (an algorithm oracle) tells you that there is no polynomial time algorithm to solve the 3Partition problem. In this part of the problem we will explore the relationships between 2Partition, 3Partition, and previously undiscussed 12Partition. In the left column of the table below, we have listed a sequence of polynomial time reductions from an instance of problem A to an instance of problem B, using symbols: $A \leq_P B$. In the right column, select the statement that most accurately reflects the significant, new information established by the reduction. Don't forget to use 1) the assumption that 3Partition has no polynomial time algorithm, and 2) the insight into 2Partition you developed in the previous part of the quiz.

$A \leq_P B$	Answer
12P*artition* $\leq_P$ 3P*artition*	◯ A has a polynomial time algorithm. ◯ B has a polynomial time algorithm. ◯ A has no polynomial time algorithm. ◯ B has no polynomial time algorithm. ◯ None of these statements is significant new information established by the reduction.
3P*artition* $\leq_P$ 12P*artition*	◯ A has a polynomial time algorithm. ◯ B has a polynomial time algorithm. ◯ A has no polynomial time algorithm. ◯ B has no polynomial time algorithm. ◯ None of these statements is significant new information established by the reduction.
12P*artition* $\leq_P$ 2P*artition*	◯ A has a polynomial time algorithm. ◯ B has a polynomial time algorithm. ◯ A has no polynomial time algorithm. ◯ B has no polynomial time algorithm. ◯ None of these statements is significant new information established by the reduction.
2P*artition* $\leq_P$ 12P*artition*	◯ A has a polynomial time algorithm. ◯ B has a polynomial time algorithm. ◯ A has no polynomial time algorithm. ◯ B has no polynomial time algorithm. ◯ None of these statements is significant new information established by the reduction.

1. Assume, that in addition to knowing that 3Partition is not in P, we know that 12Partition is in NP-Hard. In the table above, circle the reduction(s) that cannot occur unless $P = NP$.

2. In the table above, place an asterisk (*) next to the reduction(s) that we can use to show that 12Partition is in NP-Hard.

4.1 Quiz Solution

Group Pre-Quiz:

1. Partition the following set of 12 integers into 6 sets of size 2, all 6 of which have the same set-sum. While you do this simple task, try to envision an algorithm by which it can be done.

$$\{78, 187, 178, 22, 54, 13, 141, 72, 146, 122, 59, 128\}$$

$$\{78, 122\}, \{187, 13\}, \{178, 22\}, \{54, 146\}, \{141, 59\}, \{72, 128\}$$

2. Give an instance of 3Partition with $n = 4$, for which 3Partition returns YES.

$$\{78, 122, 20, 187, 22, 11, 54, 141, 25, 59, 72, 89\}$$

3. Provide evidence that your instance from the previous part results in YES by listing the subsets (i.e., the certificate).

$$\{78, 122, 20\}, \{187, 22, 11\}, \{54, 141, 25\}, \{59, 72, 89\}$$

Individual Quiz:

1. Give a brief step-by-step description of an algorithm to solve the 2Partition problem.

   - $target = \frac{1}{n}$(set-sum of S). ($\Theta(n)$)
   - Sort S. ($\Theta(n \log n)$)
   - $\forall i \in [0..n)$ pair up $S[i]$ and $S[2n - i]$. ($\Theta(n)$)
   - Scan to check set sums. If any set-sum is not equal $target$, return NO. Else return YES. ($\Theta(n)$)

2. What is the asymptotic running time of your algorithm?

   Answer: See annotations above. Total running time is $\Theta(n \log n)$, assuming merge sort.

Individual Quiz, Next page:

Suppose a reliable authority figure (an algorithm oracle) tells you that there is no polynomial time algorithm to solve the 3Partition problem. In this part of the problem we will explore the relationships between 2Partition, 3Partition, and previously undiscussed 12Partition. In the left column of the table below, we have listed a sequence of polynomial time reductions from an instance of problem A to an instance of problem B, using symbols: $A \leq_P B$. In the right column, select the statement that most accurately reflects the significant, new information established by the reduction. Don't forget to use 1) the assumption that 3Partition has no polynomial time algorithm, and 2) the insight into 2Partition you developed in the previous part of the quiz.

$A \leq_P B$	Answer
12P*artition* $\leq_P$ 3P*artition*	○ A has a polynomial time algorithm. ○ B has a polynomial time algorithm. ○ A has no polynomial time algorithm. ○ B has no polynomial time algorithm. ● None of these statements is significant new information established by the reduction.
3P*artition* $\leq_P$ 12P*artition*	○ A has a polynomial time algorithm. ○ B has a polynomial time algorithm. ○ A has no polynomial time algorithm. ● B has no polynomial time algorithm. ○ None of these statements is significant new information established by the reduction.
12P*artition* $\leq_P$ 2P*artition*	● A has a polynomial time algorithm. ○ B has a polynomial time algorithm. ○ A has no polynomial time algorithm. ○ B has no polynomial time algorithm. ○ None of these statements is significant new information established by the reduction.
2P*artition* $\leq_P$ 12P*artition*	○ A has a polynomial time algorithm. ○ B has a polynomial time algorithm. ○ A has no polynomial time algorithm. ○ B has no polynomial time algorithm. ● None of these statements is significant new information established by the reduction.

1. Assume, that in addition to knowing that 3Partition is not in P, we show that 12Partition is in NP-Hard. In the table above, circle the reduction(s) that cannot occur unless $P = NP$.

   Answer: 12P*artition* $\leq_P$ 2P*artition*

2. In the table above, place an asterisk (*) next to the reduction(s) that we can use to show that 12Partition is in NP-Hard.

   Answer: 3P*artition* $\leq_P$ 12P*artition*

4.2 Assignment

Prove that 12Partition is NP-Complete:

1. Write pseudocode for an algorithm that takes as input a set of positive integers S of size $2n$, and a

partition of S into n subsets of size 12 and returns TRUE if the set-sum of all 12 sets is the same, and FALSE otherwise.

2. What is the running time of your algorithm in the previous part in terms of n?
3. **Is 12Partition $\in NP$?**
4. Reduce 3Partition to 12Partition. That is, describe an algorithm that takes as input an instance x of 3Partition, and returns an instance y of 12Partition so that x has a valid partition into sets of size 3 if and only if y has a valid partition into sets of size 12.
5. What is the running time of your reduction in terms of n?
6. How much space does your reduction use in terms of n?
7. **Is this a polynomial reduction?**
8. Prove that your reduction from 3Partition to 12Partition is correct.
9. **Make a closing statement about 12Partition.**