CS 341, Winter 2019
Trevor Brown, Semih Salihoglu, Doug Stinson

Assignment 4 (due Friday, March 22nd, 6:00pm)

Instructions:

- Hand in your assignment using Crowdmark. Detailed instructions are on the course website.
- Give complete legible solutions to all questions.
- Your answers will be marked for clarity as well as correctness.
- For any algorithm you present, you should justify its correctness (if it is not obvious) and analyze the complexity.

1. Suppose we have a set X of n pairs of numbers (a_i, b_i), such that for each i the sum of $a_i + b_i = m$. Our goal is to find out if it is possible to divide X into two disjoint subsets of size $n/2$, S and $R = X - S$, such that the average of the a values in both S and R are strictly greater than $m/2$ (or equivalently the sum of the a values is greater than $mn/4$).

   Example. Consider the following example with $n = 4$ pairs of numbers where $m = 200$.

	a	b
p_1	110	90
p_2	84	116
p_3	120	80
p_4	94	106

   The answer to this question is YES because if we set $S = \{p_1, p_4\}$ and $R = \{p_2, p_3\}$, then the a values in S is $204 > 200$. Similarly the a values in R is $204 > 200$.

   (a) Define the following subproblems ($i = 0, \ldots, n$, $j = 0, \ldots, n$, $\ell = 0, \ldots, mn$):

   $$C[i, j, \ell] = \begin{cases} 1 & \text{if there exists a subset } S \subseteq \{1, \ldots, i\} \text{ such that } |S| = j \text{ and } \sum_{k \in S} a_k = \ell \\ 0 & \text{else} \end{cases}$$

   First give a dynamic programming algorithm to compute $C[i, j, \ell]$ for all i, j, ℓ. Remember to derive the base cases and a recurrence formula, provide justification for your formula, and write and analyze your pseudocode. The running time and space should be polynomial in n and m.

   Solution: Throughout this solution the value 0 we use in the solution array stands for "false" and 1 for "true".

   Base cases. $C[0, j, \ell] = 0$ for each j and ℓ, except $C[i, 0, 0] = 1$. This follows since an empty subset results in a sum of 0.

Recurrence.

$$C[i,j,\ell] = C[i-1,j,\ell] \vee C[i-1,j-1,\ell-a_i].$$

(For $\ell < a_i$, we take $C[i,j,\ell] = C[i-1,j,\ell]$; $C[i-1,j-1,\ell-a_i]$ will access a negative cell and we assume is 0.)

The recurrence follows from the observation that we can construct a subset S out of $a_1, ..., a_i$ of size j that sum to ℓ either:

- S is a subset of size j that sums to ℓ using $a_1, ..., a_{i-1}$. This is the first part of the recurrence.
- Or there is an S' that uses $a_1, ..., a_{i-1}$ and sums to $\ell - a_i$ and we put a_i into S' to construct S.

DP(X):

1. $C[n, n, mn/4]$ solution array (note the 2nd index can go up to $n/2$ only);
2. $C[i, 0, 0] = 1$ for each i
3. $C[0, j, \ell] = 0$ for each j and ℓ
4. for $i = 1 \ldots n$:
5. for $j = 1 \ldots n$:
6. for $\ell = 1 \ldots mn$:
7. $C[i,j,\ell] = C[i-1,j,\ell] \vee C[i-1,j-1,\ell-a_i]$.
8. return C (whether or not the answer is YES will be answered in the next part)

The runtime is $O(mn^3)$.

(b) Now using the table $C[\cdot,\cdot,\cdot]$ computed in part (a), solve the problem from part (a), i.e., decide whether there exists a subset $S \subset \{1, \ldots, n\}$ of size $n/2$ such that the average of $\{a_k : k \in S\}$ is more than $m/2$ and the average of $\{a_k : k \in R\}$ is also more than $m/2$. The running time and space should be polynomial in n and m.

Solution: Let $\sum_{i=1\ldots n} a_i = A$. We need to check if there is a $k > mn/4$ and $A - k \geq mn/4$ such that there is a subset of size $n/2$ using $a_1, \ldots, a_n$ that sum to k.

9. for $k = mn/4 \ldots A - mn/4$: ;
10. if $C[n, n/2, k]$:
11. return true;
12: return false

(c) Write pseudocode to recover a subset S with the stated property in (b) if it exists, and analyze the running time.

Solution: We write a backtracking algorithm to construct S. We first find the k that returns *true* in part (b). We then backtrack one at a time from a_n back to a_1 to see if each one is in S or not.

```
1.  remSum=k; //remainder sum left in S
2.  remItems=n/2; //remainder number of items left in S
3.  S = {}
4.  for i = n ... 1:
5.      if C[i − 1, remItems − 1, remSum − a_i] == 1:
6.          S.add(a_i); remItems = remItems − 1; remSum = remSum − a_i
7:  return S
```

2. Given a sequence of n integers $a_1, \ldots, a_n$ and an integer t, we want to divide the sequence into t chunks so as to minimize the sum of the squares of the *chunk sums*. In other words, we want to find $1 < i_1 < i_2 < \ldots < i_{t-1} < n$ to minimize $(a_1 + \ldots + a_{i_1-1})^2 + (a_{i_1} + \ldots + a_{i_2-1})^2 + \ldots + (a_{i_{t-1}} + \ldots + a_n)^2$. Present an $O(tn^2)$-time dynamic programming algorithm to solve the problem. Give pseudocode to compute the minimum value.

   [Hint: define a class of $O(nt)$ subproblems. . .]

   Solution: Define the following subproblem for each $0 \le i \le n$ and $0 \le j \le t$: let $A[i, j]$ be the minimum sum of the squares of the chunk sums over all possible division of $a_1, ..., a_i$ into j chunks. In other words, $A[i, j]$ is the minimum of the expression $(a_1 + ... + a_{i_1-1})^2 + (a_{i_1} + ... + a_{i_2-1})^2 + \ldots + (a_{i_{j-1}} + ... + a_i)^2$ over all $1 < i_1 < i_2 < \ldots < i_{j-1} < i$. If we compute all entries in A, the results is $A[n, t]$.

   Base Cases: $A[0, 0] = 0$, $A[0, j] = \infty$ for all $j \ge 1$. We assume empty chunks are not allowed.

   Recurrence: Suppose in the optimal solution to $A[i, j]$ the last chunk is $(a_{\ell+1}, \ldots, a_i)$ for some $\ell < i$. Then $A[i, j]$ would be equal to the square of this chunk plus the optimal solution to $A[\ell, j - 1]$, i.e., the minimum way to chunk $a_1, \ldots, a_\ell$ into $j - 1$ chunks. So we can write $A[i, j]$ as:

$$A[i, j] = \min_{\ell=0,\ldots,i-1} [A[\ell, j - 1] + (a_{\ell+1}, \ldots, a_i)^2]$$

   In our implementation below to save time, we will not compute $(a_{\ell+1}, \ldots, a_i)^2$ by summing over $a_{\ell+1}, \ldots, a_i$. Instead, we compute the prefix sums $s_i = a_1, \ldots, a_i$ first. Given the prefix sums we can compute $(a_{\ell+1}, \ldots, a_i)^2$ in constant time as $(s_i - s_\ell)^2$.

```
1.  DP(a_1, ..., a_n):
2.  A[n, t] solution array;
3.  A[0, 0] = 0, A[0, j] = ∞ for all j ≥ 1
4.  for i = 1, ..., n:
5.      for j = 1 ..., t:
6.          A[i, j] = ∞;
7.          for ℓ = 0 ... i − 1:
8.              if A[ℓ, j − 1] + (s_i − s_ℓ)^2 < A[i, j]:
9.                  A[i, j] = A[ℓ, j − 1] + (s_i − s_ℓ)^2;
10. return A[n, t]
```

   Runtime of the algorithm is $O(n^2 t)$.

3. Given a directed graph $G(V, E)$ with k *strongly connected components*, describe a $O((n+m)k)$-time strongly connected components algorithm that only uses BFS traversals. Solutions that use DFS will not get any credit.

   [Hint: Think of the definition of what an SCC is and how to find the SCC of a particular vertex v. Your solution might require modifying the graph G along the way.]

   Solution: By definition, the SCC of a node v contains exactly the set of nodes that v can reach and can be reachable from. If there are k $SCCs$ in G, we can find each SCC iteratively as follows. Let G^T be the transpose of the graph. We give a high level psuedocode:

   1. for $i = 1, \ldots k$:
   2. Take a node v in G whose SCC has not yet been found.
   3. run BFS in G from v and label those nodes reached by BFS
   4. run BFS in G^T from v and label those nodes reached by BFS
   5. SCC_v is the set of nodes that were labeled twice.
   6. Remove SCC_v from G.

   Each iteration of the algorithm can be implemented in $O(n + m)$ time: the BFSs will take $O(n+m)$ time and removing the SCC_v can be done in $O(n+m)$ time as well. (you don't have to elaborate on this in your answers but one way to do this is as follows: have an array of size n and label the vertices in SCC_v with a flag; then loop through each vertex u's adjacency list in G, suppose stored in a linked list, and remove each neighbor of u if it's in SCC_v). This will happen k times, each one takes $O(n + m)$ time.

4. Given n intervals $[a_1, b_1], \ldots, [a_n, b_n]$ where each interval $[a_i, b_i]$ has weight w_i, we want to find a subset of intervals whose union covers $[0, 1]$ while minimizing the total weight.

   Example: if $[a_1, b_1] = [-0.2, 0.6], w_1 = 1, [a_2, b_2] = [-0.1, 1.1], w_2 = 3, [a_3, b_3] = [0.5, 1.2], w_3 = 1$, then we would pick the first and third intervals, with total weight 2. [Note: this weighted problem does not appear to be solvable by a greedy approach.]

   Show that this problem can be reduced to finding the shortest path between two fixed vertices in a certain weighted directed acyclic graph (DAG) with n vertices and $O(n^2)$ edges. Hence, design a $O(n^2)$ time reduction algorithm that uses as a subroutine an algorithm from class.
   Solution: We construct a weighted directed graph $G(V, E)$ where $V = \{1, \ldots, n, s, t\}$ and s and t are two special vertices. We place an edge (i, j) in E of weights w_j iff:

$$a_j \leq b_i < b_j$$

   In addition, we place an edge (s, i) of weight w_j iff $a_i \leq 0 \leq b_i$; we place an edge (i, t) of weight 0 iff $a_i \leq 1 \leq b_i$. This graph G will have $O(n)$ vertices and $m = O(n^2)$ edges. In addition, G will be acyclic since $(i, j) \in E$ implies $b_i < b_j$ (so (j, i) cannot exist) and if we sort the nodes according to their b values and put s to the beginning and t to the end, we would get a topological order of G.

   Now we find the shortest (s, t) path in G, which can be done in $O(n + m) = O(n^2)$ time given the single source shortest paths algorithm for DAGs from class. We prove next that

the vertices on this path give the indices of the intervals of an optimal solution to the given covering problem.

Proof of Correctness: We show a one-to-one correspondence between the $s \rightsquigarrow t$ paths in G and intervals covering $[0, 1]$. This suffices because the shortest path in G then corresponds to the subset with the minimum weight that covers $[0, 1]$.

$\rightarrow$: Consider a path $s, i_1, \ldots, i_\ell, t$ in G. Then (s, i_1), $(i_2, i_3), \ldots, (i_\ell, t) \in E$, i.e., 0 is contained in $[a_{i_1}, b_{i_1}]$, b_{i_1} is contained in $[a_{i_1}, b_{i_2}]$ ... and 1 is contained in $[a_{i_\ell}, b_{i_\ell})$. Therefore the union of $[a_{i_1}, b_{i_1}]$ ... $[a_{i_\ell}, b_{i_\ell})$ covers $[0, 1]$.

$\leftarrow$: Next, consider a set S of intervals, whose union covers $[0, 1]$. Then 0 must be covered by some $[a_{i_1}, b_{i_1}] \in S$, b_{i_1} must be covered by some $[a_{i_2}, b_{i_2}] \in S$ (with $b_{i_2} > b_{i_1}$), b_{i_2} must be covered by some $[a_{i_3}, b_{i_3}] \in S$ (with $b_{i_3} > b_{i_2}$), ... until we reach an interval $[a_{i_z}, b_{i_z})$ that covers 1. It then follows that (s, i_1), $(i_2, i_3), \ldots, (i_z, t) \in E$ is an $s \rightsquigarrow t$ path in G.

5. Consider a connected graph $G = (V, E)$ with n vertices and positive edge weights $w_e > 0$ on the edges $e \in E$. Assume that the w_e's are *distinct* in G. Let $T = (V, E')$ be a spanning tree of G. The *bottleneck edge* of T is the edge in E' with the highest edge weight. A spanning tree T_b of G is a *minimum bottleneck spanning* (MBST) tree of G is there is no other spanning tree T' of G with a smaller bottleneck.

   (a) Prove that every minimum spanning tree (MST) of G is also an MBST.

   Solution: Suppose that T is a minimum spanning tree of $G = (V, E)$, and T' is a spanning tree with a lighter bottleneck edge. Thus, T contains an edge e that is heavier than every edge in T'. Notice that removing e from T will disconnect T into two connected components, S and $V - S$. Now consider the cut $(S, V - S)$ in T'. Because T' is spanning, there is an edge e' that crosses this cut with $w(e') < w(e)$. Consider the tree $T'' = T - \{e\} \cup \{e'\}$. Notice that T'' is indeed a tree, i.e., it is connected and acyclic: (1) Connectedness: In $T - \{e\}$, the only sets of vertices that cannot reach each other are U and V. Both S and $V - S$ are connected internally. So adding e' to $T - \{e\}$ will connect S and $V - S$, so will span V. (2) Acyclicity: Adding e' cannot create a cycle in T because it is the only edge crossing $(S, V - S)$: Recall if e' were part of a cycle then there would need to be at least one other edge crossing $(S, V - S)$.

   Notice however now $w(T'') < w(T)$, contradicting the fact that T was an MST. Therefore every MST is also an MBST.

   (b) Give a counterexample to the claim that an MBST is always an MST.

   Solution: Consider a graph G with vertices $V = \{a, b, c, d\}$ and edges $E = \{ab, bc, bd, cd\}$ and weights $w(ab) = 5$, $w(bc) = 1$, $w(bd) = 2$, and $w(cd) = 3$. Then $\{ab, bd, cd\}$ is a MBST but is not an MST. (The MST is $\{ab, bc, bd\}$.)