Solution to HW 2

CS 473: Algorithms, Fall 2018

(due Wednesday, at noon, September 12, 2018)

Version: **1.01**

---

1 (100 PTS.) Weighted discrepancy on the line.

You are given a set $W = \{w_1, \ldots, w_n\}$ of n numbers, where $w_i \in [\![U]\!] = \{1, \ldots, U\}$, for all i, where U is some positive integer number. The purpose of this question is to partition the numbers into two sets that are as balanced as possible. In particular, let $s_i \in \{-1, +1\}$ be an indicator to which of the two sets w_i is assigned to, for all i. An ***assignment*** as such, is a vector $S \in \{-1, +1\}^n$, where the two resulting sets in the partition are $L(S) = \{w_i \mid s_i = -1\}$ and $R(S) = \{w_i \mid s_i = 1\}$.

1.A. (50 PTS.) Given an assignment $S = (s_1, \ldots, s_n)$, the ***balance*** of the partition is

$$B(S) = \left| \sum_{i=1}^{n} s_i w_i \right| = \left| \sum_{x \in L(S)} x - \sum_{y \in R(S)} y \right|$$

Provide an algorithm, as fast as possible, that computes (and outputs) the assignment that realizes

$$\mathcal{B}(w_1, \ldots, w_n) = \min_{S \in \{-1,+1\}^n} B(S).$$

The running time would depend on both n and U. What is the running time of your algorithm?

How much space does your algorithm need? The smaller, the better.

Can you improve the space, if you only need to compute the value of the optimal solution (not the assignment itself).

Pseudo-code for the algorithm is highly recommended for this part.

Solution:

Let $U' = \sum_{i=1}^{n} w_i = O(nU)$. Let $f(i, x)$ be true, if there is an assignment $s_1, \ldots, s_i \in \{-1, +1\}$ for the first i numbers, such that $\sum_{\ell=1}^{i} s_i w_i = x$. Let $f(i, x)$ be a function that is 1 (or true if you prefer) $\iff$ the number x can be written as a sum (with signs) that adds up to x. Observe that if $f(i, x) = f(i, -x)$, since we can flip all the signs.

Let $-W = \{-w \mid w \in W\}$. Observe that

$$f(i, x) = \begin{cases} f(i, -x) & x < 0 \\ 1 & i = 1 \text{ and } x \in (W \cup -W) \\ 0 & i = 1 \text{ and } x \notin (W \cup -W) \\ 1 & i > 1 \text{ and } \Big(f(i-1, x - w_i) \text{ or } f(i-1, x + w_i)\Big) \\ 0 & \text{otherwise.} \end{cases}$$

is 1 (or true if you prefer) $\iff$ the number x can be written as a sum (with signs) that adds up to x. We are interested in computing the first x in $0, 1, \ldots, U$ such that $f(n, x)$ is true.

Throwing in memoization, we have that there are two parameters, which means that overall the size of the memoization table is $O(U'n) = O(n^2 U)$. We are interested in computing $f(n, 0), f(n, 1), \ldots, f(n, U)$. Which takes $O(nU') = O(n^2 U)$ time, since filling an entry takes $O(1)$ time.

To improve the space, observe that every row requires only the previous row, as such we only need to remember the previous row. In addition, we only need to remember the last value we used to get a specific value

```
register(p, prev_loc, w, i)
  if M[p,i] = 1 then return
  M[p,i] ← 1
  prev[p,i] ← prev_loc
  val[p,i] ← w
```

```
Print(p, i)
  if i = 0 then return
  print val[p,i]
  return Print(prev[p,i], i − 1)
```

```
PrintSolution:
  For ℓ = 0 to U′ do
    if M[ℓ,n] = 1 then
      Print(ℓ,n)
      return
```

```
FillIt({w_1,...,w_n})
  U′ = 2(∑_{i=1}^{n} |w_i|).
  M[−U′,...,U′,0,...,n] ← 0.
  prev[−U′,...,U′,0,...,n] ← null.
  val[−U′,...,U′,0,...,n] ← null.
  M[−U′,...,U′,,0,...,n] ← 0.
  M[0,0] ← 1.
  for i = 1 to n do
    for ℓ = −U′ to U′ do
      if M[ℓ,i−1] = 1 then
        register(ℓ − w_i, ℓ, −w_i, i)
        register(ℓ + w_i, ℓ, w_i, i)
```

Observe, that if we only care about if a solution exists, then we do not need the whole table – we just need the previous row in the dynamic programming, and the space can be reduced to $O(U')$.

1.B. (50 PTS.) Given a parameter k, let $\mathcal{S}(k,n)$ be the set of all vectors in $\{-1,+1\}^n$ that have at most k coordinates with -1 in them. Provide an algorithm, as fast as possible, that computes the assignment that realizes $\min_{S\in\mathcal{S}(k,n)} B(S)$.

Solution:

We define a similar function to part (A), here the evaluation is done in order.

$$g(i,x,k') = \begin{cases} 0 & k' < 0 \\ 1 & i = 1 \text{ and } x \in W \\ 1 & i = 1 \text{ and } x \in -W \text{ and } k' > 0 \\ 0 & i = 1 \\ 1 & i > 1 \text{ and } \Big(g(i-1, x+w_i, k'-1) \text{ or } g(i-1, x-w_i, k')\Big) \\ 0 & \text{otherwise.} \end{cases}$$

As before, we need to compute $g(n,v,k)$, for $v = 0,\ldots,U$ (we want to find the first one that is 1). Using memoization, there are $O(U'nk) = O(n^3U)$ different distinct calls, and each one takes $O(1)$ time. As such, we get an algorithm with running time $O(n^3U)$.

1.C. (Not for submission, but you can think about it.)

Given integers $i \le j$, and an assignment S, let $B(S,i,j) = \left|\sum_{\ell=i}^{j} s_i w_i\right|$ be the balance of the subsequence $w_i,\ldots,w_j$ for the given assignment. The ***discrepancy*** of the input, given an assignment S, is $D(S) = \max_{i\le j} B(S,i,j)$. The discrepancy of the input $\mathcal{D}(w_1,\ldots,w_n) = \min_{S\in\{-1,+1\}^n} D(S)$. Provide an algorithm, as fast as possible, that computes the assignment that realizes $\mathcal{D}(w_1,\ldots,w_n)$.

Solution:

Yeh, not for you and also not for me.

2 (100 PTS.) Good path.

(To solve this problem, you might want to revisit topological ordering, and how to compute it in linear time.)

Consider a DAG G with n vertices and m edges that represents courses available. Each vertex v of G corresponds to a course, with value α_v (which might be negative, if it is a useless course). A vertex v is ***useful*** if $\alpha_v > 0$.

2.A. (20 PTS.) Show an algorithm that in linear time computes all the vertices that can reach a sink of G via a path that goes through at least one useful vertex.

Solution:

Compute a topological ordering of the vertices of G: $v_1, \ldots, v_n$, where v_n is a sink. Mark all the vertices that are useful as $p(v) = 1$, and $p(v) = 0$ for all other vertices.

Now, consider the vertices $v_n, v_{n-1}, \ldots, v_1$ in this ordering. In the ith iteration, handling vertex $x = v_{n-i+1}$, consider all its outgoing edges $(x, y) \in \mathsf{E}(\mathsf{G})$. Clearly, y was already handled by this algorithm. If $p(y) = 1$ then set $p(x) = 1$.

Clearly, this algorithm runs in linear time, and we claim that it computes for each vertex if there exists a useful path. To this end, consider a vertex v_i of the graph. If $p(v_i) = 1$ by the end of the execution of the algorithm then clearly v_i can reach a useful vertex, and we are done.

If v_i can reach a useful vertex z, then consider the directed path π from v_i to z in the graph. By easy induction on π it is now easy to argue that $p(x) = 1$ for all the vertices of π, which implies that $p(v_i) = 1$.

2.B. (20 PTS.) Describe an algorithm that, in linear time, computes all the vertices that can reach a sink of G via a path that goes through at least β useful vertices, where β is a prespecified parameter.

Solution:

Same algorithm as before, except that we will have a new array $d[\cdot]$. Where initially we set:

(i) $d(x) \leftarrow 1$ if x is a sink and $\alpha_x > 0$,
(ii) $d(x) \leftarrow 0$ if x is a sink and $\alpha_x \leq 0$, and
(iii) $d(x) \leftarrow -\infty$, otherwise.

Now, when handling the vertex $x = v_{n-i+1}$, an edge $(x, y) \in \mathsf{E}(\mathsf{G})$, we set

$$d(x) = \max(d(x), d(y) + [\alpha_x > 0]),$$

where $[\alpha_x > 0]$ is one if the $\alpha_x > 0$ and 0 otherwise. The same argumentation as above now works – if $d(x) \geq \beta$ in the end of the execution of the algorithm, then x has a path to a sink with at least β useful vertices on it.

2.C. (30 PTS.) Show an algorithm, as fast as possible, that computes for all the vertices v in G the most beneficial path from v to any sink of G. The ***benefit*** of a path is the total sum of the values of vertices along the path.

Solution:

Same algorithm as before. For a vertex $x \in \mathsf{V}(\mathsf{G})$, set $d(x) \leftarrow \alpha_x$ if x is a sink, and set $d(x) \leftarrow -\infty$ otherwise.

Now, when handling the vertex $x = v_{n-i+1}$, and an edge $(x, y) \in \mathsf{E}(\mathsf{G})$, we set

$$d(x) = \max\Big(d(x), \alpha_x + d(y)\Big)$$

2.D. (30 PTS.) Using the above, describe how to compute, in linear time, a path that visits all the vertices of G if such a path exists.

Solution:

Set the value of all the vertices in the given DAG to 1. Compute the maximum profit path starting from each vertex, using the algorithm above. Now check if any vertex has benefit n.

3 (100 PTS.) Frequent flier miles.

You are give an directed graph G with n vertices and m edges, positive prices on the edges of G, a parameter $k \leq n$, and two vertices s and t. (That is, for every edge $(u, v) \in \mathsf{E}(\mathsf{G})$, there is an associated price $p\big((u, v)\big) > 0$.) Describe an algorithm, as fast as possible, that computes the shortest path from s to t, where the path is allowed to not pay for (at most) k edges used by the path. What is the running time of your algorithm? (You can use the Dijkstra algorithm as a subroutine if you want/need to.)

Solution:

Oh, this is fun - one can do this using dynamic programming, but it is in easier to build the state space of this problem. Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$. Let H be a graph, where $\mathsf{V}(\mathsf{H}) = \{(v, i) \mid v \in \mathsf{V}, i \in \{0, 1, \ldots, k\}\}$. The edges of the graph are the following:

$$\mathsf{E}(\mathsf{H}) = \left\{ \big((u,i),(v,i)\big) \;\middle|\; (u,v) \in \mathsf{E}, i = 0, \ldots, k \right\} \cup \left\{ \big((u,i),(v,i-1)\big) \;\middle|\; uv \in \mathsf{E} \right\}$$
$$\cup \left\{ \big((u,i-1),(u,i)\big) \;\middle|\; u \in \mathsf{V} \right\}$$

The price of an edge $((u, i), (v, i))$ in this graph is $p\big((u, v)\big)$, for all i, and all $(u, v) \in \mathsf{E}$. The price of all other edges is zero. Clearly, the shortest path between $(s, 0)$ and (t, k) is the desired path. This path can be computed, using Dijkstra, in

$$O\big(|\mathsf{V}(\mathsf{H})| \log |\mathsf{V}(\mathsf{H})| + \mathsf{E}(\mathsf{H})\big) = O(nk \log nk + km + kn) = O(nk \log n + km),$$

since we can safely assume $k \leq n - 1$.