CS 580 — Algorithm Design, Analysis, And Implementation
Fall 2021 — Vassilis Zikas — HW 3

Due Friday Oct. 15 at 11:59 p.m.

Important Note: *You are **only allowed one late day** for this assignment i.e. last date to submit the assignment (with late penalty) is 11:59 pm Monday, October 18. In particular, the deadline to submit without any penalty is still 11:59 pm Friday, October 15, and any submission made after 11:59 pm Friday, October 15 and before 11:59 pm Monday, October 18 will receive a 10% reduction in the grade.* **No submissions will be accepted/graded after 11:59 pm Monday, October 18!**

1. (30 points, 10 + 20)

   (a) You are given an undirected graph G with c connected components and positive edge weights. Consider the set S of subgraphs [1] of G such that any element H of S has $n-c$ edges and the property that if any two vertices are connected in G then they are also connected in H. Find and analyze an efficient algorithm that finds an element (subgraph of G) of S with maximum sum of weights of edges.

   (b) Janet was given a square matrix of size $n \times n$ with entries 0's and 1's, the goal was to come up with an efficient divide and conquer algorithm to find the length of the longest sequence of 1's in the matrix. The sequence can be horizontal or vertical, in other words, it can be contiguous sequence of 1's in a row or in a column. Note that a combination of both horizontal and vertical is NOT valid, for example, L shaped figures are not to be considered. Help Janet design and analyze an efficient divide and conquer algorithm, perhaps as fast as $O(n^2)$, that finds the longest sequence of 1's.

Answer:

(a) First, observe that any spanning subgraph of G with $n-c$ edges must be a spanning forest. To find the maximum weight spanning forest in S, we multiple the edge weights by -1 and find the minimum spanning forest in the transformed graph using Kruskal's algorithm. Time complexity is $O(\sum_{i=1}^{c} n_i \log c_i) = O(n \log n)$ where $n_1, n_2, \ldots n_c$ are vertex sizes of the c connected components. Correctness follows from Kruskal's algorithm.

We now need to show that a spanning graph with n - c edges must be a spanning forest. We can do induction on c. Consider the base case, $c = 1$. It is easy to verify that spanning subgraph with $n-1$ edges must be a spanning tree (can be verified using induction on n).

Suppose that the statement is true for all graphs on n vertices and $c-1$ components. We need to prove the statement for a graph G on n vertices, c components, and $n-c$ edges. Consider any two components of

[1] Recall that a graph H is called a subgraph of graph G if $V(G) = V(H)$ and $E(H) \subseteq E(G)$. For instance, consider the graph G on 4 vertices, $V(G) = \{a,b,c,d\}$ and edge set $E(G) = \{(a,b),(b,c),(c,d),(d,a)\}$. Then the graph H with the vertex set $V(H) = \{a,b,c,d\}$ and edge set $E(H) = \{(a,b),(c,d)\}$ is a subgraph of G. However, the graph H' with the vertex set $V(H') = \{a,b,c\}$ is not a subgraph of G because the vertex set of H' and G are different.

G, and add an edge between vertices in the two components. The resulting graph G' has $c-1$ components and $n-c+1 = n-(c-1)$ edges. By the inductive hypothesis G' must be a spanning forest i.e. acyclic. Now remove the edge we added from G'. The remaining graph must also be acyclic since removing an edge cannot introduce cycles i.e. G is a spanning forest. Hence proved.

(b) Here is a divide and conquer algorithm to find the a longest sequence of 1's in the matrix.

Divide Step: Divide matrix into 4 quadrants each having side of length $\frac{n}{2}$ as shown below. Recursively find the longest sequences in each quadrant: S_1, S_2, S_3, S_4.

1	**2**
4	**3**

Conquer Step: Recursively solve subproblems.

Combine Step:

Consider quadrant-1. At each recursive step we store lengths of longest sequence of 1's anchored at each side of the quadrant.

Let S_{ij} denote the length of 1's that go across the quadrants i and j. We combine right anchored sequences of quadrant-1 with the left anchored sequences of quadrant-2 to get S_{12}. With the stored information this takes $\theta(n)$ time.

Similarly we can find S_{23}, S_{34}, S_{14}. At the end to find the length of longest sequence in the whole matrix of size n is a trivial max that takes constant time.

Furthermore, we need to propagate the information regarding longest sequence of 1's anchored at any side to further recursive steps. Let us again consider the quadrants 1 and 2. Suppose we need to find longest right anchored sequence of 1's when we combine these quadrants. There are two options possible.

- Longest right anchored sequence of 1's in right quadrant is less than $\frac{n}{2}$, combined longest sequence will have same length as this.
- Longest right anchored sequence of 1's in right quadrant involves the whole row i.e it has length $\frac{n}{2}$, combined longest sequence would have length $\frac{n}{2}$ + length of longest right anchored sequence in left quadrant.

This information can be propagated in $\Theta(n)$ time. Thus the combine step in total is $\Theta(n)$.

The recursive relation for the algorithm is given by

$$T(n^2) = 4T(n^2/4) + \Theta(n)$$

Substitute $m = n^2$

$$T(m) = 4T(m/4) + \Theta(\sqrt{m})$$

Applying master theorem we get

$$T(m) = \Theta(m) \implies T(n^2) = \Theta(n^2).$$

Other correct solutions will also be awarded points.

2. (40 points, 5 + 5 + 15 + 15 respectively) Mario is playing a game that is modeled as a directed graph, $G : (V, E)$. Each node $u \in V$ is associated with a positive value L_u which denotes the number of lives Mario gains when she reaches that vertex. Each edge $e \in E$ is associated with a negative cost C_e which denotes the lives required

to traverse the edge. Mario can traverse an edge only if she has enough lives to do it. She starts the game at level $V_s \in V$ and needs to reach $V_t \in V$ with maximum possible lives.

For example, if Mario is at node $u \in V$ with 50 lives (this includes lives collected at node u), and $C_{u,v} = -10$, then Mario will have 40 lives on reaching v (this excludes the lives she collects from v) if she chooses to traverse edge (u, v). If $C_{u,v} = -55$, then she can not traverse the edge because she only has 50 lives. Note that she gets the lives at node v only after reaching it.

We are also guaranteed that there will be no directed cycles in the graph.

(a) First, consider the following case. $L_{V_s} = E_0$ i.e. Mario collects E_0 lives at the first node itself, however, for all other vertices $u \in V \setminus V_s$, $L_u = 0$ and Mario only has to spend 1 life to go between any two nodes (i.e. $C_{u,v} = -1$ for all u and v with an edge from u to v). Briefly describe an efficient algorithm that returns the optimal (maximal) number of lives that Mario will have when reaching node V_t. Give the time and space complexity of your algorithm.

(b) Now, Mario spends different amounts of lives to go from one node to another. $L_{V_s} = E_0$ i.e. Mario collects E_0 lives at the first node itself, however, for all other vertices $u \in V \setminus V_s$, $L_u = 0$. Briefly describe an efficient algorithm that returns the optimal (maximal) number of lives that Mario will have when reaching node V_t. Give the time and space complexity of your algorithm.

(c) Now, lives at each node could also be positive. Describe an efficient algorithm that returns the optimal (maximal) number of lives that Mario will have when reaching node V_t. Give the time and space complexity of your algorithm.

Hint: Maybe there is a way to define the problem in terms of subproblems.

(d) This time, the game is exactly the same as the previous version - negative edge costs and non-negative vertex lives - except Mario begins the game with a single skip superpower. She can use the superpower at any time to jump to any node she wants (including nodes already traversed, except for the final node V_t) and recollect the lives at any nodes she reaches after using the superpower. Modify your previous algorithm to account for this case. Give the time and space complexity of your algorithm.

Answer: V stands for number of vertices and E for number of edges in the graph.

(a) Edge weights are equal. Run a BFS algorithm from V_s as source and V_t as target. If the shortest path has life cost E' then lives at V_t is $E_0 - E'$. Mario can only reach the V_t if $E_0 > E'$.
Time Complexity: $O(V + E)$.
Space Complexity: $O(V + E)$.
Correctness follows from shortest path algorithm.

(b) Multiply all edge weights by -1 to make them positive. We can run Dijkstra's algorithm and get the minimum path length to be E'. Lives at V_t is $E_0 - E'$. Time Complexity: $O(E + V\log(V))$ using Fibonacci heaps.
Since our modified graph is a DAG we can use topological sorting to find the shortest path. Time required for this is $O(V + E)$. Bellman-Ford as a solution only gets partial credit.

(c) Let $OPT(v)$ be the optimum lives achieved till node v starting from V_s with E_0 lives. The recurrence is given by:

$$OPT(v) = \left\{ \begin{array}{ll} E_0 & \text{If } v = V_s \\ max_{u \in Inc(v), OPT(u)+C_{u,v}>0}\{OPT(u) + C_{u,v} + E_v\}, & \text{Otherwise} \end{array} \right\}$$

Note that $Inc(v)$ denotes the incoming neighborhood of vertex v. Time complexity is $O(V + E)$ (sum of degrees of all vertices in a graph is $O(E)$). The space complexity is $O(V)$.

You can apply the Bellman Ford's algorithm on a modified graph. This gives partial credit because of worse runtime.

Let vertices in the topological ordering be $V_1, V_2 \dots V_n$. Let V_s be the first vertex in the ordering. We can show correctness of the recursion using induction.

Base case is $n = 1$ i.e. $V_1 = V_s$. Base case is true by definition.

Let the inductive hypothesis be that $OPT(V_i)$ is calculated correctly for all values of $i < k$ i.e. $OPT(V_i)$ gives the optimum lives achieved till node V_i starting from V_s.

Consider the node V_k. Because the graph is acyclic, the only paths from V_s to V_k are through vertices in $\{V_2 \dots V_{k-1}\}$. Consider the optimal path from V_s to V_k that achieves the maximum number of lives at V_k. Let it be denoted by $\{V_s, V_{i_1} \dots V_{i_l}, V_k\}$, where $V_{i_l} \in Inc(V_k)$. This path can be decomposed into the optimal path from V_s to V_{i_l} and the edge (V_{i_l}, V_k). Therefore, the number of lives at V_k using this path is $OPT(V_{i_k}) + C_{V_{i_l},V_k} + E_{V_k}$ by the inductive hypothesis. In the recursive step, we consider the maximum lives that we can have over all the incoming edges. Note that we only consider those incoming neighbors for which we have enough lives to traverse the incoming edge based on the optimal lives at the incoming vertex.

(d) Let $OPT(v,s)$ be the optimum lives achieved till node v starting from V_s with E_0 lives and s skips remaining. s can be either 0 or 1. The recurrence is given by:

$$OPT(v,s) = \left\{ \begin{array}{ll} E_0 & \text{If } V = V_s, s = 1 \\ max_{u \in Inc(v), OPT(u,s)+C_{u,v}>0}\{OPT(u,s) + C_{u,v} + E_v\}, & s = 1 \\ max \left\{ \begin{array}{ll} max_{u \in Inc(v), OPT(u,s)+C_{u,v}>0}\{OPT(u,0) + C_{u,v} + E_v\} & (Eq^nA) \\ max_{u \in (V')}\{OPT(u,1) + E_v\} & (Eq^nB) \end{array} \right\}, & s = 0 \end{array} \right\}$$

V' gives the set of vertices in the graph.

Number of subproblems: $O(V)$. Time complexity = $O(V^2)$. Space complexity $O(V)$.

Correctness can be shown same as the previous case using induction. Here, we give a brief justification of each case in the recursion.

The base case is trivially true.

$s = 1$ implies that the skip superpower has not been used yet, In this case, the recursion is exactly like the previous part.

When $s = 0$, there are two options, either the super power was not used in the latest step (Eq^nA) or it was used in the latest step (Eq^nB). The correctness for the equation A can be argued similar to the previous part. In equation B, the path looks like $\{V_s, V_{i_1} \dots V_{i_l}, (skip), V_k\}$ which can be decomposed into optimal path from V_s to V_{i_l} without any skips used and one skip power used to jump to V_k. We recurse over all vertices in the graphs in this recursive case.

3. (30 points) Consider the following "Balls and Bins" problem. You are given n pairs of balls (i.e. $2n$ balls in total), denoted by $\{A_{11}, A_{12}, A_{21}, A_{22}, \dots A_{n1}, A_{n2}\}$ where $\{A_{i1}, A_{i2}\}$ represent a pair, and m buckets, denoted by $\{B_1, B_2, \dots B_m\}$. We want to allocate balls to buckets. The goal is to maximize the total number of balls assigned to some bucket (some balls will be left un-assigned) such that the following conditions are satisfied.

- For each ball, say A_{ij} where $i \in \{1, 2, \dots, n\}, j \in \{1, 2\}$, there is a list of buckets associated, denoted by P_{ij}. A_{ij} can be assigned to a bucket B_k only when $B_k \in P_{ij}$.
- Bucket B_i can take at most b_i balls.
- For balls that form a pair i.e. for any i, the balls $\{A_{i1}, A_{i2}\}$ cannot be assigned to the same bucket.
- Each ball goes into at most one bucket.

Design and analyze an efficient algorithm to find the maximum number of balls that can be assigned.

Answer:

Solution. We formulate the problem as a max-flow problem on a graph G. G is constructed as follows:

- We first define the vertex set of G. We first add a source node S and a sink node T. In addition to the source S and sink T we add nodes $x_{11}, \ldots, x_{n2}$ (one for each ball), $y_1, \ldots, y_m$ (one for each bucket). If any pair of balls say A_{i1}, A_{i2} can be assigned to the same bucket say B_j then we add an extra node C_{ij} i.e. for all integral $i \leq n$ and integral $j \leq m$, if $B_j \in P_{i1}$ and $B_j \in P_{i2}$, then we create the extra node denoted by C_{ij}.
- Connect a node source S to nodes for all balls (one for each ball) with edges of capacity 1 i.e., for each x_i we add the directed edge (S, x_i) with capacity $c(S, x_i) = 1$.
- Connect a node target T to node y_j with capacity b_j. We do this for all $j \leq m$.
- For each ball A_{ij} add edges of capacity 1 from x_{ij} to all buckets that the ball can be assigned to but its pair cannot be assigned to i.e. we add the edges (x_{ij}, y_j) of capacity 1 if $B_j \in P_{ij}$ and $B_j \notin P_{i\bar{j}}$, where $\bar{j} = |1 - j|$ i.e. if $j = 0$, then $\bar{j} = 1$ and if $j = 1$, then $\bar{j} = 0$.
- For each node of the type C_{ij} add the edges $(x_{i1}, c_{ij}), (x_{i2}, c_{ij})$, and (c_{ij}, y_j) each of capacity 1. The node c_{ij} has capacity 1 to ensure that both balls A_{i1} and A_{i2} are not assigned to the same bucket y_j.

Run max-flow algorithm on graph G. The value of max-flow gives the maximum total balls that can be assigned.

Time Complexity Analysis:
The graph above has $E = O(n + m + nm)$ edges. The max flow in the graph can be atmost $min\{\sum_{j=1}^{m} M_j, 2n\}$.
Let the max-flow be f. If we use Ford-Fulkerson on G the runtime would be $O(Ef)$.
Graph construction also takes $O(n + m + nm)$ time.

To argue correctness, we will prove that the value of the max flow is indeed equal to the maximum number of balls that can be assigned. Note that the flow given by Ford-Fulkerson will be integral. Because of the graph construction the max flow in the graph dictates a valid assignment of balls. Therefore, number of balls that can be assigned is atleast value of max flow. Suppose that for purpose on contradiction, more balls can be assigned. Each balls assignment corresponds to a S-T path in the constructed graph. The union of these paths is a valid flow. Therefore, max flow is atleast the number of balls that can be assigned.

4. **2 points** Have you assigned pages on Gradescope? **Answer:** Yes!