CS5800 Algorithms

Recitations from Nov 11, 2019 to Nov 15 2019

Ravi Sundaram

Problem 1 (Network Flow)

1. Compute the maximum flow of the following two flow network using the Edmonds-Karp algorithm, where the source is S and the sink is T. Show the residual network after each flow augmentation. Write down the maximum flow value.

$$\begin{tikzcd}
S \arrow[r, "2"] \arrow[d, "5"'] & C \arrow[r, "2"] \arrow[dr, "4"] & D \arrow[r, "2"] & T \\
A \arrow[r, "6"] & B \arrow[u, "1"] \arrow[r, "3"] \arrow[ur, bend right, "2"'] & E \arrow[ur, "10"] &
\end{tikzcd}$$

2. There are N filled water tanks in county A and $3M$ empty water tanks in county B. The i-th water tank in county A currently holds A_i gallons of water. The j-th empty water tank in county B has a capacity of B_j gallons of water.

   The city has decided to remove all the water from the tanks in county A for repairs and transport the water to county B. The water in any tank in county A can be transported to any tank in county B.

   After some time, the tanks in county B will be emptied to fill 3 new pools. However,

   - Pool 1 can be filled only by tanks $1, 2..., M$ in county B
   - Pool 2 can be filled only by tanks $M + 1, M + 2..., 2M$ in county B
   - Pool 3 can be filled only by tanks $2M + 1, 2M + 2..., 3M$ in county B

   Since any water that cannot be poured into a pool will be discarded, you have to design an algorithm to find the maximum total amount of water that can be filled in all the pools at the end.

3. You are working for Bluebikes and you need to figure out whether the bikes can be transported around (so that popular stations have bikes supplied to them). At some given moment, let $d(v)$ denote the number of additional bikes needed at station v. This can be less than 0, in which that station can give up that many bikes to a different station. And for some pairs of stations, u, v,

there is a van available to transport $c(u, v)$ bikes from u to v. Give an algorithm to determine whether the vans can tranport bikes around to satisfy the demand of every station.

Solution:

1. The augmenting paths at each iteration (and the amount of flow is)

$$SCDT - 2 \tag{1}$$
$$SABET - 3 \tag{2}$$
$$SABCET - 1 \tag{3}$$
$$SABDCET - 1 \tag{4}$$

2. We create the vertices s, t. Then for each tank i in county A we create the vertex a_i and the edge (s, a_i) with capacity A_i. Then for each tank j in county B we create the vertex b_j. Between all pairs of (a_i, b_j) we create the edge (a_i, b_j) with capacity A_i. Next, we create the vertices p_1, p_2, p_3 for each pool. For p_1 we create the edges (b_1, p_1) with capacity B_1, (b_2, p_1) with capacity B_2 ,..., (b_m, p_1) with capacity B_m. We do similar things with p_2, p_3 and their respective tanks. Finally, we create the edge (p_1, t) with capacity $\sum_{i=1}^{M} B_i$, the edge (p_2, t) with capacity $\sum_{i=M+1}^{2M} B_i$, and the edge (p_3, t) with capacity $\sum_{i=2M+1}^{3M} B_i$.

   Now we run max flow on the above graph. The max flow corresponds to the max amount of water that we can put in the three pools. The capacity for each edge (s, a_i) corresponds to the fact that pool a_i holds A_i gallons of water. Then we send all the water in county A to the tanks in county B. Then the capacity of edge (b_j, p_i) corresponds to the fact that tank j can hold at most B_j gallons of water. The other capacities are just the maximum amount of water that can go through that step, but don't actually limit anything.

3. For each station we create a vertex v with demand $d(v)$. We can map this as a flow problem following the standard reduction from circulation to network flow. We create the vertices s, t. For each vertex v with $d(v) < 0$ we add the edge (s, v) with capacity $-d(v)$. For each vertex with $d(v)$ we add the edge (t, v). For each van between stations u, v, we add the edge (u, v) with capacity $c(u, v)$. Now we run max flow and if its equal to $\sum_{v:d(v)>0} d(v)$ then there is a valid circulation. Otherwise there isn't.

...

■

Problem 2 (Dynamic Programming)

1. A special rod-cutting machine enables a worker to break a rod into two pieces. To cut a rod of length l in any two pieces costs l. Suppose a worker wants to break a rod into many pieces by cutting it at different break points. The order in which the breaks occur can affect the total amount of time used.

   For example, suppose that the worker wants to break a rod of length 20 into 4 pieces by cutting it at lengths 2, 8, and 10 from one end of the rod. If they perform the breaks in the order just given, then the first break costs 20, the second 18, and the third costs 12, totaling 50. However, if they perform the breaks in the opposite order, the first break costs 20, the second costs 10, and the third 8, totaling 38.

Design an algorithm that, given a rod R of length l and an array $L[1...m]$ containing the break points, computes the lowest cost for a sequence of breaks.

Solution:

1. - English Description of problem: $OPT[i, j]$ will contain the penalty of the optimal sequence of breaks of the subrod that starts at breakpoint i and ends at breakpoint j.
   - English Description of logic: When we are choosing a break point $i < k < j$, the penalty is always the same at the the current iteration. The penalty is just the length of the subrod which we can express as $L[j] - L[i]$. To make the first and last break points work, we set $L[0] = 0$ and $L(m + 1) = l$. For each breakpoint k, the remaining penalty can be expressed as $OPT[i, k] + OPT[k, j]$. So we want the minimum sum among all ks.
   - Recurrence Relation (and Base Case)

$$OPT[i,j] = \begin{cases} 0 & \text{if } i+1=j \\ L[j] - L[i] + \min_{\substack{k \text{ s.t.} \\ i<k<j}} (OPT[i,k] + OPT[k,j]) & \end{cases}$$

   - Iterative Algorithm - The outer loop will be over i, from m to 1, and the inner look will be over j, from $j = i + 1$ to $j = m + 1$ starting at $i = m + 1$ up to $i = n$.
   - Number of subproblems -There are $O(m^2)$ subproblems.
   - Time per subproblem - We need to find the min over all breakpoints. There are in the worst case takes linear time, so $O(m)$.
   - Total Runtime - We get $O(m^3)$.
   - Final Return Value - $OPT[0, m + 1]$

...

■

Problem 3 (Extras...)

1. Suppose you are given two arrays $A[1...m], B[1...n]$. Give a dynamic programming solution to compute the length of the shortest common supersequence of A and B. A supersequence is a string that contains both A and B as subsequences. For example if we are given the words card and creed, then the shortest common supersequence is careed. Hint: You can solve this using Longest Common Subsequence. You can also solve it without. Try doing it both ways.

2. This is a problem we've seen before. You are about to go on a journey, and you are thinking of what to take in your bag (of limited size). You are given a set of n items. Each item has a value v_i and a weight w_i that are always in $\{0, 1, 2, ..., n\}$. You are given a target value V and a target weight W. Describe a polynomial time algorithm that determines if there exists a subset S of the the items such that $\sum_{i \in S} v_i \geq V$ and $\sum_{i \in S} w_i \leq W$.

   Now suppose we have the additional constraint that you can bring at most k items in your bag. Give a solution for this variant.