CSC 373 - Algorithm Design, Analysis, and Complexity **Summer 2016**

Assignment 2: Due Tuesday July 5, 10PM

Please follow the instructions provided on the course website to submit your assignment. You may submit the assignments in pairs. Also, if you use **any** sources (textbooks, online notes, friends) please cite them for your own safety.
You can use those data-structures and algorithms discussed in CSC263 (e.g. merge-sort, heaps, etc.) and in the lectures by stating their name. You do not need to provide any explanation or pseudo-code for their implementation. You can also use their running time without proving them: for example, if you are using the merge-sort in your algorithm you can simply state that merge-sort's running time is $\mathcal{O}(n \log n)$.

Every time you are asked to design an efficient algorithm, you should provide both a short high level explanation of how your algorithm works in plain English, and the pseudo-code of your algorithm similar to what we've seen in class. State the running time of your algorithm with a brief argument supporting your claim. You must prove that your algorithm finds an optimal solution!

1 Fight Fight Fight ... Or come up with an algorithm.

You're at a fair with a group of friends, when you saw a booth selling a booklet of tickets to a Eurocup match. Excited, you decided to check it out. Unfortunately with all your coins combined, you didn't have enough to buy the booklet. The vendor challenged you[1] to play a game and if you win, you win the booklet. You won!

Once the booklet in hand, you realized it didn't have enough tickets for all of you to go. Luckily [2], there is an even number of you n, and the booklet has $n/2$ tickets. You decided the fairest way to get tickets is to split into two groups, and play the following game, and the team to win the most money gets the booklet.

The game goes as follows: You each put a coin down. The coins are placed in a row. Your team goes first. At every step of the game you can take a coin from an end point of the row. The other team goes next and does the same. You keep alternating until no coins are left. The ordering of the coins is arbitrary, and the coins don't necessarily have equal value. Give an algorithm that allows your team to win as much as possible. Prove it is correct.

Solution:

Since the opponent is as smart as us, we need to take all possible combinations in consideration before our move. Calculating all these moves will include repetitive calculations, but DP is here to save the day :)

Suppose the coins are stored in an array A. Let $S(i, j)$ be the maximum sum possible when the only remaining coins are from i to j. We have two cases to consider when it's our turn: We can either choose coin $A[i]$ or coin $A[j]$.

Case 1 - Suppose we pick coin $A[i]$: Then the opponent has a choice between $A[i+1]$ and $A[j]$. If the opponent takes $A[i+1]$, the remaining coins are $\{A[i+2], \dots, A[j]\}$, for which our max value is denoted by $S(i+2, j)$. On the other hand, if the opponent takes $A[j]$, our max sum is $S(i+1, j-1)$. Since the opponent is as smart as us, s/he would have chosen the choice that yields the minimum amount to us.

[1] it's a fair after all
[2] or not

Therefore, the maximum amount we can get when we pick $A[i]$ is

$$C_1 = A[i] + \min\{S[i+2, j], S[i+1, j-1]\}$$

Case 2 Suppose now we pick coin $A[j]$. Using a similar reasoning to the one above, we conclude that the maximum amount we can get in this case is

$$C_2 = A[j] + \min\{S[i+1, j-1], S[i, j-2]\}$$

Therefore in order to maximize our winnings, we just pick the best out of C_1 and C_2

$$\begin{aligned} S[i,j] &= \max\{C_1, C_2\} \\ &= \max\{A[i] + \min\{S[i+2,j], S[i+1,j-1]\}, A[j] + \min\{S[i+1,j-1], S[i,j-2]\}\} \end{aligned}$$

The proof is similar to what we've seen in class. We're trying all combinations and choosing the max out of all scenarios so the solution must be optimal. I'll skip the details.

2 Thank you for your help.. Choos.

After a large scale fire that destroyed your city, the injured residents were taken to hospitals at nearby cities. You joined a volunteer organization that decided to send a group of n volunteers, including yourself, to help at these hospitals. Since this is taking place outside the city, the organizers decided to assign volunteers to hospitals within a half-hour drive from their home. The hospitals in the other hand decided to provide free meals to all the volunteers, but due to budget constraints, each hospital cannot feed more than $\lceil \frac{n}{m} \rceil$ volunteers, where m is the total number of hospitals involved in this rescuing operation.

Give an algorithm that allows the organizers to decide whether it is possible to satisfy all these constraints. Analyze its running time in term of n and m.

Solution:

Each volunteer must be sent to a hospital, so on first glance this might come across as a matching problem. However, several volunteers can be sent to one hospital, so it's not the one-to-one type of pairing that is necessary for a matching instance. But we can solve this problem using a max flow algorithm, provided we define the flow network properly, as follows:

Construct a directed bipartite graph $G(V \cup H, E)$ where V is the set of vertices representing the volunteers (one vertex per volunteer), and H the set of vertices representing the hospitals. For each volunteer, add a directed edge from that volunteer to each hospital to which s/he can get to (i.e. the set of hospitals no more than half-hour drive from the volunteer's house).

Now, to turn this directed bipartite graph to a flow network, add two vertices s and t, where for all $v \in V$ add the directed edge (s, v) and for every $h \in H$, add the directed edge (h, t). Assign a capacity of 1 to every edge in the set

$$\{(s, v) | v \in V\} \cup \{(v, h) | v \in V, h \in H\}.$$

In order to handle the meal constraint, for each edge of the form $(h, t), h \in H$, assign a capacity equal to $\lceil \frac{n}{m} \rceil$.

This leaves us with a flow network with integer capacities on the edges. We can use the Edmonds-Karp algorithm to compute a max flow. If the value of the max flow is n, then we know there is an assignment of volunteers to hospitals that satisfies all the constraints.

If the value of the max flow is less than n, then there is no satisfying assignment.

Building this graph takes time $\mathcal{O}(nm)$ time, since each of the v volunteers can be joined to at most m hospitals, and adding the additional edges to s and t takes time $\mathcal{O}(n)$ and $\mathcal{O}(m)$ respectively. So overall, the flow network takes time $\mathcal{O}(nm)$ time to construct.

Using the Edmonds-Karp algorithm, we can compute a max flow in time $\mathcal{O}((n+m+2)(nm+n+m)^2) \in \mathcal{O}((n+m)n^2m^2)$, since the Edmonds-Karp algorithm takes $\mathcal{O}(|V||E|^2)$ time, and in this case we have $|V| = n+m+2, |E| \in \mathcal{O}(nm+n+m)$.

3 More Graphs ❤

Given a graph $G(V,E)$, a vertex cover is a subset $S \subseteq V$ of vertices such that every edge $e \in E$ has an endpoint in S. We say that e is covered, hence the name vertex cover.

A k-regular graph is a graph where every vertex v has exactly k neighbours; i.e. $\forall v, |\{u|(u,v) \in E\}| = k$. We often drop the k, and call G regular.

A perfect matching is a matching where every vertex is covered by an edge.

1/ Prove that if $G(A \cup B, E)$ is bipartite, the size of the maximum matching equals the size of a minimum vertex cover.

Solution:

This theorem is known as the König-Egerváry Theorem.

Ford-Fulkerson solved this problem by reducing it to a network flow problem, and then applying the MFMC theorem.

In a bipartite graph $G(A \cup B, E)$, the size of any matching is at most the size of any vertex cover, since all matched edges must have at least one endpoint in the cover. To show that this bound is attained, we construct a flow network G' with vertices $A \cup B \cup \{s,t\}$ and edges

$$\{(s,a)|a \in A\} \cup E \cup \{(b,t)|b \in B\}.$$

The edges of E are directed from A to B in G'. We give the edges of E infinite (or sufficiently large finite) capacity, and all other edges capacity 1. Let (S,T) be a minimum s,t-cut in G'. This cut has finite capacity, since it is bounded by the capacity of the cut $\{s\}, (A \cup B \cup \{t\}) \backslash \{s\}$, which is $|A|$. Therefore, no edge in E can cross from S to T, since those edges have infinite capacity. Thus each edge in E either

- has both endpoints in S,
- has both endpoints in T, or
- crosses from T to S.

In any of these three cases, the edge is incident to an element of the set

$$(T \cap A) \cup (S \cap B),$$

so this set forms a vertex cover of G. Moreover, the capacity of the cut (S,T) is the size of this set, which is also the maximum flow value in G' by the MFMC theorem. Recall in class we showed that the size of the maximum matching in G is the value of a maximum flow in G'.

2/ Let $G(A \cup B, E)$ be a bipartite graph. For a set $S \subseteq A$, let the set $N(S)$ be the set of neighbours of S, i.e.

$$N(S) = \{v \in V | \exists u \in S, (u, v) \in E\}$$

Show that G has a matching in which every vertex of A is matched iff for every subset S of A:

$$|N(S)| \geq |S|$$

Solution:

This is *Hall's Theorem*, often called the Marriage Theorem because it tells us whether all A vertices can be "married" to a B vertex of their own.

Suppose first that G has a matching in which every vertex of A is matched. Let S be a subset of A. For every vertex $s \in S$, the vertex that s is matched to appears in $N(S)$. Since no two vertices in S are matched to the same vertex in B, we have $|N(S)| \geq |S|$.

Conversely, suppose G does not have a matching in which every vertex of A is matched. By the König-Egerváry Theorem above, there exists a vertex cover C of G containing fewer than A vertices. Let $S = A - C$. All vertices adjacent to vertices S must be in $B \cap C$, otherwise C would not be a vertex cover, i.e., $N(S) \subseteq B \cap C$. Then:

$$\begin{aligned}
|N(S)| &\leq |B \cap C| \\
&= |C| - |A \cap C| \qquad \text{since } A \text{ and } B \text{ are disjoint.} \\
&< |A| - |A \cap C| \\
&= |S| \qquad \text{since } S \text{ is defined to be } A - C.
\end{aligned}$$

Thus there exists an $S \subseteq A$ such that $|N(S)| < |S|$.

3/ Show that any non-trivial regular bipartite graph has a perfect matching.

Solution:

Let d be the degree of the vertices in the regular bipartite graph $G(A \cup B, E)$. The total number of edges is

$$d \cdot |A| = d \cdot |B|,$$

thus $|A| = |B|$; therefore any matching that uses all the vertices in A will be a perfect matching in G. By Hall's Theorem above, it suffices to show that $|S| \leq |N(S)|$ for any $S \subseteq A$. For an arbitrary subset S of A, consider the subgraph of G induced by $S \cup N(S)$. There are exactly $d \cdot |S|$ edges in this subgraph, since every vertex in S has degree d. Similarly, this subgraph has no more than $d \cdot |N(S)|$ edges, because no vertex in $N(S)$ has degree larger than d. Thus

$$d \cdot |S| \leq d \cdot |N(S)|.$$

It follows that $|S| \leq |N(S)|$.

4 Cha Cha Cha Chaaanges

Let (G, s, t, c) be a flow network with integral capacities, and let f be its maximum flow.

1/ Suppose we increase a specific capacity by 1. Show how to compute the new maximum flow in $\mathcal{O}(m)$ time, where m is the set of edges in G.

Solutions :

First thing to notice is that the value of the new max flow is either the same value of the old one, or it is incremented by at most one. This is true because the value of any min cut can only increase by one, so the value of the max flow increases by at most one.

Consider the following algorithm then: Increase the capacity of the edge, and look at the new residual graph G_f. Try to find an augmenting path in G_f, if such a path exists, augment along it. Since the capacities are integral, the flow will be augmented by one, and since this is the best we can do, we get the max flow.

2/Suppose now we decreased a capacity (≥ 1) by 1. Show how to compute the new maximum flow in $\mathcal{O}(m)$ time.

Solutions :

Similarly to the scenario above, decreasing the capacity along an edge can decrease the capacity of any cut by at most one. Therefore, the value of the max flow can decrease by at most one.

Let (u, v) be the the edge whose capacity was decremented. Given the old flow f, first decrease the flow going across it by 1. Then compute a shortest s, u path with positive flow, and decrease the flow along this path by 1. Similarly, compute a shortest u, t path with positive flow and decrease the flow along this path by 1. We now have a valid flow f' of value one less than before. Finally, try to an augmenting path in this residual graph. If no such path exists, f' is maximum, and if it exists, augment along it by at most 1 and return the max flow.

5 What Goes Up, Must Come Down... Or does it?

Let (G, s, t, c) be a flow network with vertex set V, and edge set E.

1/ Let $E^+ \subseteq E$ denote the subset of edges where for all $e \in E^+$, increasing $c(e)$ increases the maximum flow of G. Is $|E^+| > 0$ for all G? Give an algorithm that finds all $e \in E^+$, with a running time better than m max-flows.

Solutions :

The set E^+ is not always non-empty. For instance, consider the network on vertices $\{s, v, t\}$ and edges $\{(s, v), (v, t)\}$, and capacity one for all edges. Increasing the capacity of either edge does not increase the maximum flow.

If $E^+ \neq \emptyset$, we can compute it as follows:

- Compute a max flow f in the network.
- Let U be the set of all vertices reachable from s in the residual graph G_f.
- Let W be the set of vertices that can reach t in G_f.
- Any edge (u, w) with $u \in U, w \in W$ belongs to E^+.

In other words, any edge in the min cut that certifies the MFMC theorem is an edge of E^+. This is correct since increasing the capacity of any such edge results in an augmenting path (i.e. won't be in the cut).

2/ Similarly, let $E^- \subseteq E$ denote the subset of edges where for all $e \in E^-$, decreasing $c(e)$ decreases the maximum flow of G. Is $E^- = E^+$? Give an algorithm that finds all $e \in E^-$, and analyze it.

Solutions :

Notice first that the edges in the network above on $\{s, v, t\}$ are all edges of E^-. Therefore $E^+ \neq E^-$.

To compute the set E^-, consider the following lemma:

Lemma 1. *Let f be a max-flow for G, and let (u, v) be any edge in G. Consider the residual network G_f. The set of edges (u, v) such that there is no path from u to v in G_f is exactly the set E^-.*

Proof. If there is no path from u to v, then the edge (u, v) is saturated. Therefore $f(u, v) = c(u, v)$. Say we decrement the capacity by δ. Then if we are to keep the max flow at its current value, the δ units of flow over capacity must be rerouted. But note that we cannot reroute the flow because there is no path from u to v. Therefore $(u, v) \in E^-$.

Now suppose there is a path from u to v, with minimum capacity δ. If we increase the capacity of (u, v) by δ, we can reroute the flow through the path from u to v, so that the max flow would remain the same. So $(u, v) \notin E^-$. □

The algorithm is then to compute all pairs (u, v) such that (u, v) is an edge in the graph, but there is no path from u to v in the residual graph G_f. This can be done by first finding the strongly connected components (SCC) of G_f, and then the directed acyclic graph (DAG) representation of the components in $\mathcal{O}(m + n)$ time [3]. Let D_f denote the DAG of the SSC of G_f.

We perform topological sort on D_f to get T_f. Starting from root of T_f (if there are multiple roots, process them one at a time), for each node, maintain the set of vertices u such that v is reachable from u. This can be in $\mathcal{O}(m + n)$ time. Once we have this information for all the nodes recheability matrix can be done in $\mathcal{O}(n^2)$ time.

[3] Hello from 263!