The Australian National University Semester 2, 2024
School of Computing Tutorial 12

Convenor(s): Yun Kuen Cheung (Marco) & Ahad N. Zehmakan

Concrete version: Sunday 20th October, 2024, 15:09 (Compilation date and time)

Algorithms (COMP3600/6466)

Problems marked with $(*)$ are challenge exercises. They will not be discussed in tutorials, and solutions will not be released. Once confident in your solution for a challenge exercise, you are welcome to discuss it with your tutor during the consultation period in the tutorial session, or schedule a time with Marco to present it.

Exercise 1 *Activity Selection Problem*

1. In the lecture, we present a greedy algorithm for the activity selection problem, which runs in $\mathcal{O}(n^2)$ time. Actually, by some simple modifications of the algorithm, we can improve it to $\mathcal{O}(n \log n)$ time. The algorithm presented in lecture maintains an array of activities sorted by their finishing times. Now, simultaneously, we also maintain an array of activities sorted by their starting times. Explain how the new array helps with reducing the runtime to $\mathcal{O}(n \log n)$.

Solution

In the new array, the activities are sorted by their starting times. Thus, when running the algorithm presented in class, when an activity with finishing time t is selected, we can use binary search on the new array to identify, among the remaining activities, the activity with the latest starting time while overlapping with the just-selected activity. Say this latest starting time be s^*. All activities with starting time less than or equal to s^* are then eliminated.

In each iteration, the binary search takes $\mathcal{O}(\log n)$ time. As there are at most n iterations, the overall runtime for all binary searches is $\mathcal{O}(n \log n)$. It is possible that in one round many activities are eliminated, but notice that throughout the whole algorithm, each activity is eliminated at most once, so the overall runtime for all eliminations is bounded by $\mathcal{O}(n)$.

2. Now, suppose each activity has a possibly distinct value, and the problem becomes to select non-overlapping activities that maximize the sum of their values. Construct a problem instance to show that the greedy algorithm presented in lecture does not compute the optimal solution.

Solution

Suppose there are three activities. The first activity has starting time 1 and finishing time 3, and its value is 1. The second activity has starting time 4 and finishing time 6, and its value is 1. The third activity has starting time 2 and finishing time 5, and its value is 3. The algorithm presented in lecture selects the first two activities, achieving a total value of 2. However, the optimal solution is selecting the third activity only, achieving a total value of 3.

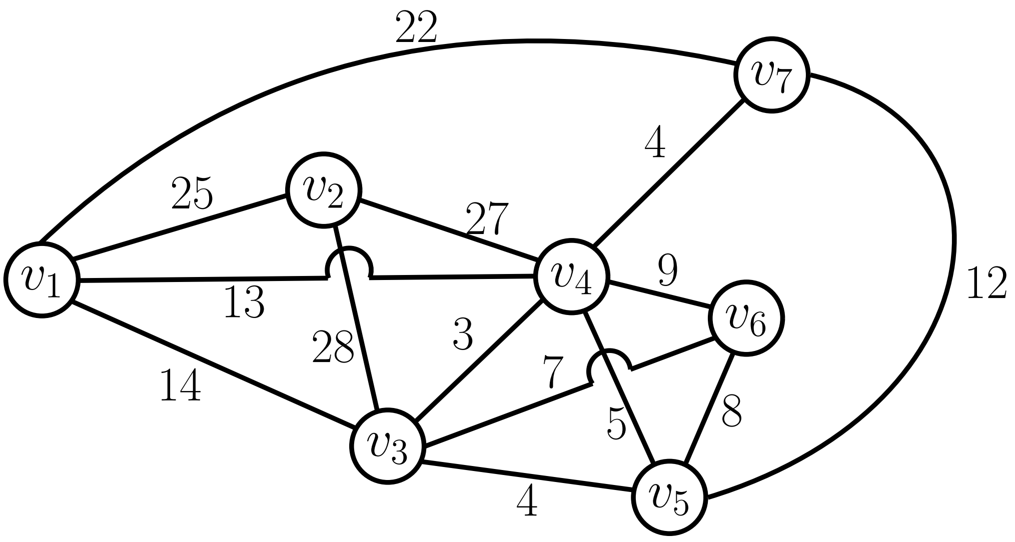
Exercise 2 *Minimal Spanning Tree Algorithms*

3. For the graph below, compute a minimum spanning tree using the following algorithms. Write down the results of intermediate steps.
   (a) Kruskal's algorithm;
   (b) Prim's algorithm with $V_H = \{v_1\}$ initially.

Solution

(a) After 1st iteration: edge $\{v_3, v_4\}$ added to H
After 2nd iteration: edge $\{v_4, v_7\}$ added to H
After 3rd iteration: edge $\{v_3, v_5\}$ added to H (the 2nd and 3rd iterations can interchange)
After 4th iteration: edge $\{v_4, v_5\}$ considered but not added to H
After 5th iteration: edge $\{v_3, v_6\}$ added to H
After 6th iteration: edge $\{v_5, v_6\}$ considered but not added to H
After 7th iteration: edge $\{v_4, v_6\}$ considered but not added to H
After 8th iteration: edge $\{v_5, v_7\}$ considered but not added to H
After 9th iteration: edge $\{v_1, v_4\}$ added to H
After 10th iteration: edge $\{v_1, v_3\}$ considered but not added to H
After 11th iteration: edge $\{v_1, v_7\}$ considered but not added to H
After 12th iteration: edge $\{v_1, v_2\}$ added to H, and we are done because H is now a spanning tree.

(b) After 1st iteration: edge $\{v_1, v_4\}$ added to H
After 2nd iteration: edge $\{v_3, v_4\}$ added to H
After 3rd iteration: edge $\{v_4, v_7\}$ added to H
After 4th iteration: edge $\{v_3, v_5\}$ added to H (the 3rd and 4th iterations can interchange)
After 5th iteration: edge $\{v_3, v_6\}$ added to H
After 6th iteration: edge $\{v_1, v_2\}$ added to H, and we are done because H is now a spanning tree.



4. In the lecture, we presented a proof that Kruskal's algorithm is correct. By using similar ideas from this proof, write a proof to show that Prim's algorithm is correct.

Solution

Suppose the contrary: T is the spanning tree generated by Prim's algorithm, but T' is a MST with strictly less weight than T. Suppose $e_1, e_2, \ldots, e_{n-1}$ are the edges added to H by Prim's algorithm in this ordering. Given T', there is an ℓ satisfying $1 \leq \ell \leq n-1$, such that T' contains edges $e_1, e_2, \ldots, e_{\ell-1}$, but it does not contain edge e_ℓ. We choose MST T' that maximizes ℓ.

Suppose $V^{\ell-1}$ is the set of visited vertices by Prim's algorithm just after the $\ell - 1$-st iteration. Now add e_ℓ to T'. There is a cycle. Since e_ℓ is an edge across $V^{\ell-1}$ and $V \setminus V^{\ell-1}$, the cycle must contain another edge which is across the two subsets too. Denote this edge by e'. Due to the design of Prim's algorithm, we have $w(e_\ell) \leq w(e')$. Then note that $T'' = (T' \setminus \{e'\}) \cup \{e_\ell\}$ is a spanning tree whose weight is at most the weight of T', and T'' contains $e_1, e_2, \ldots, e_\ell$, a contradiction.

5. This question is motivated but much simplified from a practical problem. There is no model answer, you can use your own innovation and creativity in your solution.

   A virus is spreading in a city. The local government maintains a database of patients, their home and work postcodes, and rough genetic information on the viruses extracted from their blood. The government has identified several patients as "sources", and seeks an estimate of how other patients might have contracted the virus. As an algorithm consultant, can you design an algorithm to assist with this task?

Solution

As disclaimed, there is no model answer. Below is a seemingly reasonable solution. If you think your solution is more brilliant, present it to the tutor.

First, we construct a weighted graph. Each patient is a vertex. Between every pair of patients there is an edge, and the weight of the edge represents how different the two viruses are. We impose that the weight of every edge must be strictly positive, but can be very close to zero. A weight close to zero means the two viruses are exactly the same and have common home/work postcodes, while a weight is larger if the viruses have different genetic information and different home/work postcodes.

We also add a special vertex, which has edges to the sources, and the weight of each such edge is zero. Now, we run the Prim's algorithm with V_H initially containing the special vertex. We can assure that all edges from the special vertex to the sources are added in the first few iterations.

Eventually, the algorithm returns a MST which, when you consider the special vertex as the root, every other vertex has exactly one ancestor among the sources. The path from a patient to her source ancestor suggests the chain of how she contracts her virus from the ancestor.

Exercise 3 ***Dijkstra's Algorithm***

6. For the graph given in question 3 above, use the Dijkstra's algorithm to compute the shortest path from v_1 to any other vertices. Write down the results of intermediate steps.

Solution

We list the results of each intermediate step below. (v'ed) means the vertex is already visited.

- After the 1st iteration, v_1 is visited, and we have the following table:

vertex u	v_1 (v'ed)	v_2	v_3	v_4	v_5	v_6	v_7
$D[u]$	0	25	14	13	$+\infty$	$+\infty$	22
$P[u]$	None	v_1	v_1	v_1	None	None	v_1

- After the 2nd iteration, v_4 is visited, and we have the following table:

vertex u	v_1 (v'ed)	v_2	v_3	v_4 (v'ed)	v_5	v_6	v_7
$D[u]$	0	25	14	13	18	22	17
$P[u]$	None	v_1	v_1	v_1	v_4	v_4	v_4

- After the 3rd iteration, v_3 is visited, and we have the following table:

vertex u	v_1 (v'ed)	v_2	v_3 (v'ed)	v_4 (v'ed)	v_5	v_6	v_7
$D[u]$	0	25	14	13	18	21	17
$P[u]$	None	v_1	v_1	v_1	v_4	v_3	v_4

- After the 4th iteration, v_7 is visited, and we have the following table:

vertex u	v_1 (v'ed)	v_2	v_3 (v'ed)	v_4 (v'ed)	v_5	v_6	v_7 (v'ed)
$D[u]$	0	25	14	13	18	21	17
$P[u]$	None	v_1	v_1	v_1	v_4	v_3	v_4

- After the 5th iteration, v_5 is visited, and we have the following table:

vertex u	v_1 (v'ed)	v_2	v_3 (v'ed)	v_4 (v'ed)	v_5 (v'ed)	v_6	v_7 (v'ed)
$D[u]$	0	25	14	13	18	21	17
$P[u]$	None	v_1	v_1	v_1	v_4	v_3	v_4

- After the 6th iteration, v_6 is visited, and we have the following table:

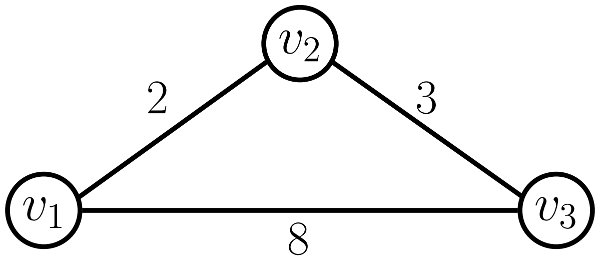
vertex u	v_1 (v'ed)	v_2	v_3 (v'ed)	v_4 (v'ed)	v_5 (v'ed)	v_6 (v'ed)	v_7 (v'ed)
$D[u]$	0	25	14	13	18	21	17
$P[u]$	None	v_1	v_1	v_1	v_4	v_3	v_4

- After the 7th and final iteration, v_2 is visited, and we have the following table:

vertex u	v_1 (v'ed)	v_2 (v'ed)	v_3 (v'ed)	v_4 (v'ed)	v_5 (v'ed)	v_6 (v'ed)	v_7 (v'ed)
$D[u]$	0	25	14	13	18	21	17
$P[u]$	None	v_1	v_1	v_1	v_4	v_3	v_4

7. You are given a graph $G = (V, E, w)$ of traffic network, where $w(e)$ is the petroleum cost to travel from one endpoint of e to the other endpoint. Furthermore, there is a transit cost function $c : V \to \mathbb{R}_+$. For any vertex v, $c(v)$ is a cost we need to pay if we travel via v but v is not our destination.

   For instance, in the graph below, suppose $c(v_1) = 4$, $c(v_2) = 5$ and $c_3 = 10$. If we want to go from v_1 to v_3, the cost of using the lower path is $c(v_1) + 8 = 12$, while the cost of the using the upper path (via v_2) is $c(v_1) + c(v_2) + 2 + 3 = 14$.



   Design an algorithm that, given the graph G, the transit cost function c, and the source vertex v, computes the least-cost path to go from v to any other vertices. Analyze its running time.

Solution

The problem can be reduced to a standard single-source shortest path problem on a new *directed* graph.

Given the input graph $G = (V, E, w)$, to construct the new graph, for each vertex $u \in V$, break it into two vertices u_{in} and u_{out}. We add a directed edge from u_{in} to u_{out}, whose weight is $c(u)$. For each edge $\{u, x\} \in E$, we construct two directed edges, one from x_{out} to u_{in}, and the other from u_{out} to x_{in}. Both directed edges have weight $w(\{u, x\})$.

One crucial observation is, for any path $v \to u^1 \to u^2 \to \ldots \to u$ from the source v to some other vertex u in the original graph, its cost (including the transit costs) is exactly equal to the length of the path $v_{in} \to v_{out} \to u^1_{in} \to u^1_{out} \to u^2_{in} \to u^2_{out} \to \ldots \to u_{in}$ in the new graph.

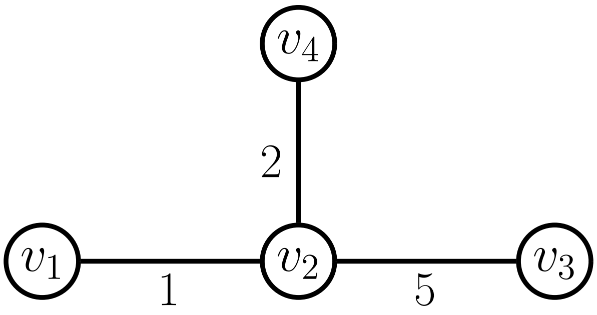
Another crucial observation is, for any vertex u_{in} in the new graph, its only outgoing edge is to u_{out}, and for any vertex u_{out} in the new graph, all its outgoing edges are towards in-vertices. Thus, any path in the new graph must be alternating between in-vertices and out-vertices, and if u_{in} is in the path but not the final vertex, then its next vertex must be u_{out}.

In sum of the previous paragraphs, for any $u \neq v$, the shortest path from v_{in} to u_{in} in the new graph must also correspond to the least-cost path from v to u in the original graph. Thus, applying the Dijkstra's algorithm to the new graph can compute the least-cost paths in the original graph from v to any other vertices.

The new graph has $2|V|$ vertices and $2|E| + |V|$ directed edges. The runtime using min-heap is $\mathcal{O}((2|E| + |V|) \log(2|V|)) = \mathcal{O}((|E| + |V|) \log |V|)$.

8. (*) You are given a graph $G = (V, E, w)$ of traffic network, where $w(e)$ is the petroleum cost to travel from one endpoint of e to the other endpoint. Furthermore, there is a reward function $r : V \to \mathbb{R}_+$. For any vertex v, $r(v)$ is the amount of money we receive if we visit v.

   For instance, in the graph below, suppose $r(v_1) = 1$, $r(v_4) = 7$, and $r(v_2) = r(v_3) = 0$. Then the least-cost path to go from v_1 to v_3 is $v_1 \to v_2 \to v_4 \to v_2 \to v_3$, and its cost is $1+2+2+5-r(v_1)-r(v_4) = 2$. In comparison, the path $v_1 \to v_2 \to v_3$ incurs a cost of $1+5-r(v_1) = 5$. Also, we note that the least-cost path to go from v_1 to itself is $v_1 \to v_2 \to v_4 \to v_2 \to v_1$, and its cost is $1+2+2+1-r(v_1)-r(v_4) = -2$.



   The problem is, given the graph G, the reward function r, and the source vertex v, computes the least-cost path to go from v to any other vertices. Note that here a path from v to some vertex u may visit some vertex multiple times — in particular, u can be visited multiple times. However, we can only get reward from a vertex once even if we visit it multiple times.

   (a) Either design a polynomial-time algorithm for the problem, or prove that the problem is NP-hard.

   (b) Suppose now we have the restriction that G is a tree. Can you design a polynomial-time algorithm for the problem?