Pre-tutorial questions

Do you know the basic concepts of this week's lecture content? These questions are only to test yourself. They will not be explicitly discussed in the tutorial, and no solutions will be given to them.

1. Greedy algorithms
   (a) What is typical with a greedy approach?
   (b) What is the Interval Scheduling problem?
   (c) What is the Interval Partitioning problem?
   (d) To prove correctness of a greedy algorithm we often use an "exchange argument". What is the idea of an exchange argument?
2. Minimum Spanning Trees
   (a) What is a minimum spanning tree (MST)?
   (b) A MST fulfills the cut property and the cycle property. Explain the two properties.
   (c) Explain the general idea of Prim's algorithm.
3. Dijkstra's algorithm
   (a) What does Dijkstra's algorithm actually compute?
   (b) State the general idea of Dijkstra's algorithm.

Tutorial

Problem 1
Given set S of n real numbers and an integer I the 3-Sum problem is to decide (true/false) if S contains three elements that sum to I. It is well known that one can solve the problem in $O(n^2)$ time. The best known lower bound for the problem is $\Omega(n \log n)$, that is, there is no algorithm that can solve the problem in less time. Given the below algorithm, can you give any upper and lower bounds on the running time of the algorithm? Assume that you have a function Decide-3-Sum(S, I) that solves 3-Sum problem for a set S and an integer I.

Algorithm 1 Print-3-Sum values

```
1: procedure Print-3-Sum values($S$,$m$)
                         ▷ $S$ is a list of real values and $m$ is an integer
2:     for $I = 1, \ldots, m$ do
3:         if Decide-3-Sum($S, I$) then
4:             Print(I)
5:         end if
6:     end for
7: end procedure
```

Solution: We don't know the implementation of DECIDE-3-SUM(S, I) we only know for sure that it requires $\Omega(n \log n)$ time, hence the algorithm has a lower bound of $\Omega(mn \log n)$ and we can't give an upper bound. However, we do know that there *exists* some algorithm that can solve the problem in $O(mn^2)$.

Problem 2
Give an $O(n)$ time algorithm to detect whether a given undirected graph contains a cycle. If the answer is yes, the algorithm should produce a cycle. (Assume adjacency list representation.)

Use DFS to find whether there is a back-edge.

There are n-1 edges in DFS tree plus a back-edge, with max height = n

Solution:
We run DFS with a minor modification. Every time we scan the neighborhood of a vertex u we check if the neighbor v has been discovered before and whether it is different than u's parent. If we can find such a vertex the we have our cycle: $v, u, \text{parent}[u], \text{parent}[\text{parent}[u]], \ldots, v$.
We only need to argue that this algorithm runs in $O(n)$ time. Consider the execution of the algorithm up the point when we discovered the cycle. After the $O(n)$ time spent initializing the arrays needed to run DFS, each call to DFS-VISIT takes time that is proportional to the edges discovered. However, up until the time we find the cycle we have only discovered tree edges. So the total number of edges is upper bounded by $n - 1$. Thus, the overall running time in $O(n)$.

Problem 3
Trace Prim's algorithm on the graph in Fig. 3 starting at node a. What's the output?

Solution: Recall Prim's algorithm.

Step 0: Set a as the root, $S = \{a\}$ and the tree $T = nil$.

Step 1: Find a lightest edge such that one endpoint is in S and the other is in $V \setminus S$. Add this edge to T and its (other) endpoint to S.

Step 2: If $V \setminus S$ is empty then stop and output the minimum spanning tree T. Otherwise go to Step 1.

The output is shown in Fig. 1.

Problem 4
Trace Dijkstra's algorithm on the graph in Fig. 3 starting at node a and ending at g. What's the shortest path?

Solution: The algorithm always choose to go to the node that is closest to the source. The output is shown in Fig. 2.

Problem 5
Let $G = (V, E)$ be an undirected graph with edges weights $w : E \to R^+$. For all $e \in E$, define $w_1(e) = \alpha w(e)$ for some $\alpha > 0$, $w_2(e) = w(e) + \beta$ for some $\beta > 0$, and $w_3(e) = w(e)^2$.

1. Suppose p is a shortest s-t path for the weights w. Is p still optimal under w_1? What about under w_2? What about under w_3?

2. Suppose T is minimum weight spanning tree for the weights w. Is T still optimal under w_1? What about under w_2? What about under w_3?

Dijkstra's Algorithm->更新每一段的距离，之后选取当前距离最小的，锁定，在此更新距离，之后重复
找最短路径，从终点开始back-trace，每一段都找距离权重最小的一个，以此类推。

Kruskal's Algorithm->Keep adding the cheapest, avoid producing cycles.

Solution:

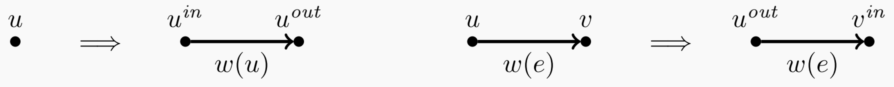
1. For w_1 we note that for any two paths p and p' if $w(p) \leq w(p')$ then
$$w_1(p) = \alpha w(p) \leq \alpha w(p') = w_1(p').$$
so if a path is shortest under w it remains shortest under w_1.

   This is now longer the case with w_2 and w_3. Consider a graph with three vertices and three edges, basically a cycle. Suppose one edge has weight 1 and the other edges have weight 1/2. Consider the shortest path between the endpoints of the edge with weight one. The two possible paths have total weight 1, so both of them are shortest. Now imagine adding 1 to all edge weights; that is, consider w_2 with $\beta = 1$. The shortest path in w_2 is unique—the edge with weight 1 in w. Now imagine squaring the edge weights; that is, consider w_3. The shortest path in w_3 is unique—the path with two edges of weight 1/2, which under w_3 have weight 1/4 each.

2. We claim that the optimal spanning tree does not change. This is because the relative order of the edge weights is the same under w, w_1, w_2, and w_3. Therefore, if we run Prim's or Kruskal's algorithm the edges will be processed in the same order and the output will be the same in all cases.

Problem 6
Consider the following generalization of the shortest path problem where in addition to edge lengths, each vertex has a cost. The objective is to find an s-t path that minimizes the total length of the edges in the path plus the cost of the vertices in the path. Design an efficient algorithm for this problem.

Solution: We reduce the problem of finding a path with minimum edges plus vertex weight to a standard shortest path problem with weights on the edges only. Given a directed graph $G = (V, E)$ and weights w we construct a new graph $G' = (V', E')$ and weights w'. For each vertex $u \in V$ we create two copies u^{in} and u^{out}, which we connect with an edge (u^{in}, u^{out}) with weight $w(u)$. For each edge $(u, v) \in E$ we create an edge (u^{out}, v^{in}) with weight $w(e)$.



It is trivial to check that for each path $p = \langle u_1, u_2, \ldots, u_k \rangle$ in the input graph G we have a path $p' = \langle u_1^{out}, u_2^{in}, u_2^{out}, \ldots, u_k^{in} \rangle$. Furthermore, the edge weight of p' equal the edges plus vertex weight of p.
It is easy to see that the graph G' can be constructed in $O(m + n)$ time. Furthermore, $|V'| = 2|V|$ and $|E'| = |V| + |E|$. Therefore Dijkstra's algorithm run in $O(m + n \log n)$, where $m = |E|$ and $n = |V|$.

Problem 7
It is not uncommon that a given optimization problem has multiple optimal solutions. For example, in an instance of the shortest s-t path problem, there could be multiple shortest paths connecting s and t. In such situations, it may be desirable to break ties in favor of a path that uses the fewest edges.

Show how to reduce this problem to a standard shortest path questions. You can assume that the edge lengths ℓ are positive integers.

1. Let us define a new edge function $\ell'(e) = M\ell(e)$ for each edge e. Show that if P and Q are two s-t paths such that $\ell(P) < \ell(Q)$ then $\ell'(Q) - \ell'(P) \geq M$.

2. Let us define a new edge function $\ell''(e) = M\ell(e) + 1$ for each edge e. Show that if P and Q are two s-t paths such that $\ell(P) = \ell(Q)$ but P uses fewer edges than Q then $\ell''(P) < \ell''(Q)$.

3. Show how to set M in the second function so that the shortest s-t path under ℓ'' is also shortest under ℓ and uses the fewest edges among all such shortest paths.

Solution:

1. Since the edge lengths are integer valued, if $\ell(P) < \ell(Q)$ then $\ell(Q) - \ell(P) \geq 1$. It follows then that $\ell'(Q) - \ell'(P) = M\ell(Q) - M\ell(P) = M(\ell(Q) - \ell(P)) \geq M$.

2. Let $|P|$ be the number of edges used in the path P. Then $\ell''(P) = M\ell(P) + |P|$. It follows that $\ell''(P) = M\ell(P) + |P| < M\ell(Q) + |Q| = \ell''(Q)$.

3. Set M to be the number of vertices in the graph. From task (b) we know that if P is optimal under ℓ'' then it will have the fewest edges among paths that have length $\ell(P)$. Now suppose that there is another path Q such that $\ell(Q) < \ell(P)$; then $\ell(P) - \ell(Q) \geq 1$ and therefore $M\ell(P) \geq M + M\ell(Q)$. Notice that since $|Q| < n$, we have

$$M + M\ell(Q) > |Q| + M\ell(Q) = \ell''(Q)$$

and also

$$M\ell(P) \leq |P| + M\ell(P) \leq \ell''(P).$$

Putting all these inequalities together we get

$$\ell''(P) \geq M\ell(P) \geq M + M\ell(Q) > \ell''(Q)$$

which contradicts the optimality of P in ℓ''.

Problem 8

Suppose we are to schedule print jobs on a printer. Each job j has associated a weight $w_j > 0$ (representing how important the job is) and a processing time t_j (representing how long the job takes). A schedule σ is an ordering of the jobs that tell the printer how to process the jobs. Let C^σ_j be the completion time of job j under the schedule σ.

Design a greedy algorithm that computes a schedule σ minimizing the sum of weighted completion times, that is, minimizing $\sum_j w_j C^\sigma_j$.

Solution: An optimal schedule can be obtained by sorting the jobs in increasing $\frac{t_j}{w_j}$ order; assume for simplicity that no two jobs have the same ratio. To show that the schedule is optimal, we use an exchange argument. Suppose the optimal solution is σ and there are two adjacent indices, say i and k such that $\frac{t_i}{w_i} > \frac{t_k}{w_k}$. We build another schedule τ where we swap the positions of i and k, and leave other jobs in their places. We need to argue that this change decreases the cost of the schedule. Notice that the completion time of jobs other than i and k is the same in σ and τ. Thus,

$$\sum_j w_j C_j^\sigma - \sum_j w_j C_j^\tau = w_i C_i^\sigma + w_k C_k^\sigma - w_i C_i^\tau - w_k C_k^\tau. \tag{1}$$

Let $X = C_i^\sigma - t_i$, the time when job i starts in σ, which equals the time when job k starts in τ. Then $C_i^\sigma = X + t_i$, $C_k^\sigma = X + t_i + t_k$, $C_k^\tau = X + t_k$, and $C_i^\sigma = X + t_i + t_k$. If we plug these values into the above equation and simplify, we get

$$\sum_j w_j C_j^\sigma - \sum_j w_j C_j^\tau = -w_i t_k + w_k t_i. \tag{2}$$

The proof is finished by noting that $\frac{t_i}{w_i} > \frac{t_k}{w_k}$ implies $-w_i t_k + w_k t_i > 0$ and therefore

$$\sum_j w_j C_j^\sigma > \sum_j w_j C_j^\tau.$$

Hence, the schedule σ is not optimal.

Problem 9

[Advanced] The k-centre problem in the Euclidean plane is defined as follows. Given a set V of n points in the Euclidean plane find a subset S of V of size k such that the maximum Euclidean distance of any point in V to its closest point in S is minimized. Consider the following trivial algorithm:

1. Pick any arbitrary point $p \in V$, set $S \leftarrow \{p\}$.
2. While $|S| < k$, find the point whose minimum distance to S is maximum and add it to S.

Prove that the above algorithm is a 2-approximation algorithm.

Solution: Let $C^* = \{c_1^*, c_2^*, \ldots, c_k^*\}$ be the cluster centres of the optimal k-clustering and let $C = \{c_1, c_2, \ldots, c_k\}$ be the cluster centres of the farthest algorithm. Set the clusters with centers in C to have radius $2Radius(C^*)$. Two cases to consider:
(a) If a cluster C_i^* contains one cluster center c_j of C then C_j includes C_i^*.
(b) If a cluster C_i^* contains zero cluster centers of C then there must exist a cluster C_j^* that includes two cluster centers of C, say c_a and c_b. This implies that no point lie further than $|c_a, c_b|$ from its closest cluster center (otherwise the algorithm wouldn't have chosen c_a and c_b). As a result $|c_a, c_b| \leq 2Radius(C^*)$.

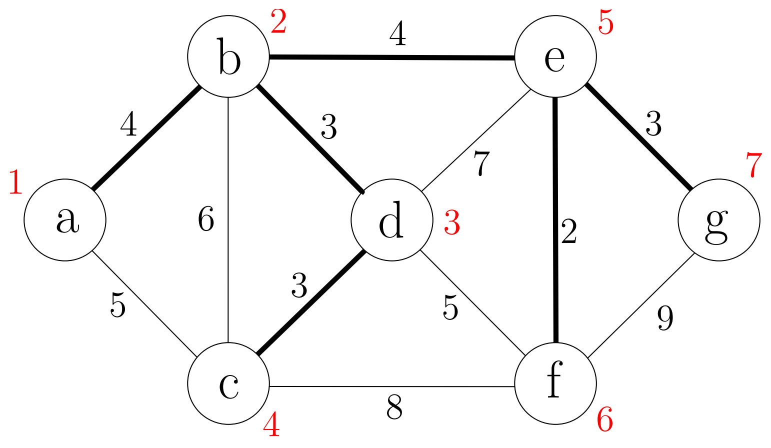


Figure 1: The red numbers show the order in which the nodes are added to the tree.

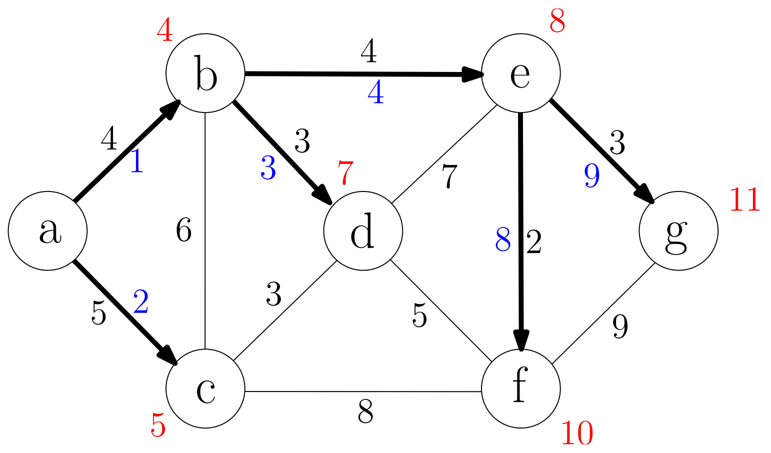


Figure 2: The red numbers show the length of the shortest path from a, and the blue numbers show the order in which the nodes are reached.

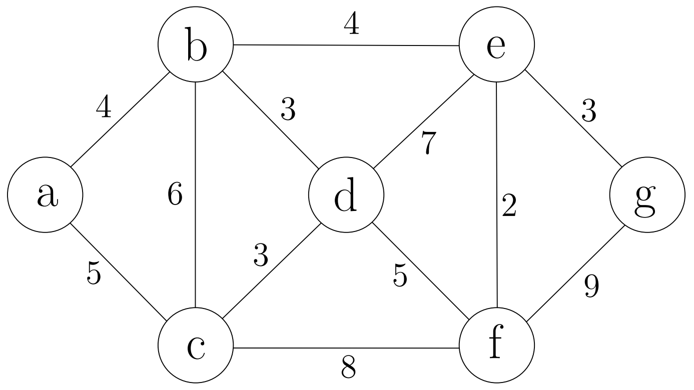


Figure 3: Input graph to Questions 3 and 4.