Cs445 — Homework #5
Due: 11/26/2015 midnight

Instructions.

1. Solution could **not** be submitted by students in pairs.
2. You could submit a pdf of the homework, either printed or hand-written and scanned, as long as it is **easily** readable.
3. If your solution is not clearly written, it might not be graded.
4. Prove the correctness of your answer. A correct answer without a proof might not be accepted.
5. If you have discussed the solution with other students, mention their names clearly on the homework. These discussions are not forbidden and are actually **encouraged**. However, you must write your whole solution yourself.
6. All questions have same weight.
7. You could refer to a data structure studied in class, and just mention briefly their guaranties. For example "*It is known that a Red-Black tree could support the insert/delete/find operations on a set of n elements in time $O(\log n)$.*
8. If your answer uses one of the data structures or algorithms that were studied in class, you could refer to it without having to repeat details studied in class. If you answer requires only modifications of one of the algorithm, it is enough to mention the required modifications, and what's the effect (if any) on the running time and on other operations that the algorithm performs.
9. In general, a complete answer should contain the following parts:
   (a) High level description of the data structures (if needed). *E.g. We use a binary balanced search tree. Each node contains, a key and pointers to its children. We augment the tree so each node also contains a field...*
   (b) High level description of the algorithms
   (c) Proof of correctness (why your algorithm provides what is required).
   (d) A claim about the running time, and a proof showing this claim.

In all questions, h is given and known to you. Typical examples are $h = 10$ or $h = 20$.

1. The question refers to a quad tree T built on a green-red image, as shown in the slide 2 and 3. All pixels of the image are taken from a $2^h \times 2^h$ integers grid, where h is an arbitrary integer. Give an example of image, for which T has 4^h leaves.

2. Consider a quadtree T , built for storing a set S of n points in the plane, all taken from the $2^h \times 2^h$ integers grid, where h is an arbitrary integer. Assume that each leaf of T contains ≤ 1 points.

   Describe the pseudo-code of a algorithm that receives a node v of T (initially $root(T)$) and a new point p, and insert p into T. The algorithm should be as efficient as possible. What is its worst running time (as a function of h) ? What is its best running time ?

   The algorithm could be recursive, but this is not a requirement.

3. Under the assumptions of question 2, suggest an input set S containing only 2 points, such that the height of the resulting quad tree T is $\geq h - 1$. Specify the coordinates of the points of S. To describe S, specify the coordinates of each point in S. E.g., if you are considering pixels on the display, the coordinates of a pixel (i, j) is its row and column.

4. The question refers to a Quadtree constructed for for storing a set of points (as discussed in the slides). Let S be a set of points, all from the $2^h \times 2^h$ grid. Let T be a quadtree built on S, as shown in the slides.

   Let Q be a query disk (given to you by its center and radius). The operation $\texttt{count}(T, Q)$ reports $|Q \cap S|$ (the number of points of S which are also inside Q).

   Suggest a slight modification of T, such that when a new query disk Q is given, you could compute $\texttt{count}(T, Q)$ efficiently.

   Write a recursive peudocode for the operation $\texttt{count}(v, Q)$, where v is a node of T. The first call to this function is with $\texttt{count}(root(v), Q)$

   Note that we are interested in *which* points of S are there in Q. Only in their number. Your code should be as efficient as possible (and would not be accepted otherwise).

5. Let $G(A \cup B, E)$ be a given undirected bipartite graph (A and B and disjoint sets of vertices, and each edge of E connects a vertex of A to a vertex of B). We say that a subset $M \subseteq E$ is called a **matching** if each vertex of V is adjacent to at most one edge of M.

   To find a maximum-cardinality matching, we convert the problem into a maximum-flow problem in the graph $G'(V', E')$, as discussed in class (and on the slides):

   (a) We set $E' = E$. We assign directions to the edges. An edge (a_i, b_j) is directed from a_i to b_j.

   (b) Add vertex s and $\forall a_i \in A$ add an edge (s, a_i).

   (c) Add vertex t and $\forall b_j \in B$ add an edge (b_j, t).

   (d) Assign capacity 1 to all edges.

As always, if a 0/1 flow is given on this network, the edges of E that carries flow are the ones in M and vice versa: If a matching $M \subseteq E$ is given, there is a single assignment of flow to the edges of G' such that M consists only of the of E that carry flow. Here is a quick reminder how: For every edge $(a_i, b_j) \in E$

$$(a_i, b_j) \in M \quad \Rightarrow$$

$$f(s, a_i) = f(a_i, b_j) = f(b_j, t) = 1 \quad ; \quad f(a_i, s) = f(b_j, a_i) = f(t, b_j) = -1$$

while

$$(a_i, b_j) \notin M \quad \Rightarrow \quad f(a_i, b_j) = 0$$

The flow across the other edges of G' are uniquely determined by these rules.

Let π be an augmenting path in G'_f. What is the minimum number of edges in π ?

You could assume that π s simple - no vertex appears twice along π

6. Under the same assumptions as previous questions - what is the maximum number of edges of π, when $|A| = |B| = n$.

7. Under the same assumptions as previous questions - assume π has k edges. So $\pi = e_1, e_2 \dots e_k$. Consider the edges of π without their direction. So if $(u, v) \in \pi$ then $(v, u) \in \pi$. (Intuitively, this means that we remove the arrow-heads from the edges, so they don't have direction anymore)

   Which edges of π are in M, and which are not in M ?

8. Give an example of a network $G(V, E)$ for which at least $2^{(n/2)-3}$ **minimum** cuts exist. What are these cuts ? Prove.

   As usual, n is an arbitrary large number.

9. Let $G(V, E)$ be a network flow. Explain how to find a maximum positive flow p in the network, such that the flow that goes through each vertex is ≤ 17.

   Formally, the flow that goes through a vertex u (which is neither s nor t) is $\sum_{v \in V} p(v, u)$.

10. Assume $G(V, E)$ is a network flow where the capacity of each edge is 1. Consider the Ford-Fulkerson Algorithm for finding a maximum flow in $G(V, E)$. As you recall, in each iteration, a new augmenting path is found, and the flow along it is augmented.

    (a) What is the maximum number of iterations (as a function of $n = |V|$ and $m = |E|$) ?

    (b) What is the minimum number of iterations (as a function of n and m) ?

    You could assume that there are no parallel edges. That is, for every $u, v \in V$, there is at most one edge connecting u to v.