CS 170 Second Midterm **ANSWERS** 7 April 2010

NAME (1 pt): ______________________________

SID (1 pt): ______________________________

TA (1 pt): ______________________________

Name of Neighbor to your left (1 pt): ______________________________

Name of Neighbor to your right (1 pt): ______________________________

**Instructions**: This is a closed book, closed calculator, closed computer, closed network, open brain exam, but you are permited a 1 page, double-sided set of notes, large enough to read without a magnifying glass.

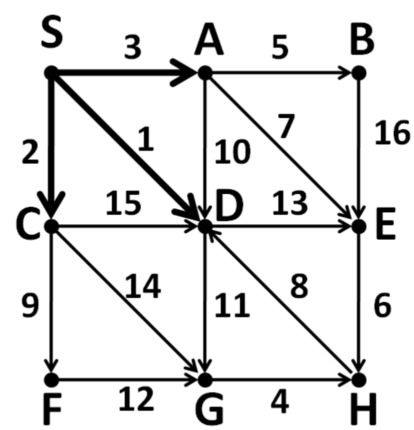
You get one point each for filling in the 5 lines at the top of this page. Each other question is worth 20 points. No points will be subtracted for wrong answers so it's in your best interest to guess all you want.

Write all your answers on this exam. If you need scratch paper, ask for it, write your name on each sheet, and attach it when you turn it in (we have a stapler).

| | |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| Total | |

**Question 1 (20 points).** We are running the following four algorithms on the graph below, where the algorithms have already "processed" the three bold-face edges:

- Dijkstra's algorithm for shortest paths, starting from $S$.
- Prim's algorithm for the Minimum Spanning Tree (MST), starting from $S$ (ignoring edge directions).
- Kruskal's algorithm for the Minimum Spanning Tree (MST) (ignoring edge directions).
- Breadth-First-Search (BFS) starting from $S$ (ignoring both edge directions and edge weights, but visiting neighboring vertices in lexicographic order).




**(a)** Which 3 edges would be added next to the MST in Prim's algorithm? Be sure to indicate the order in which they are added.

Answer: First (A,B), then (A,E), then (E,H).

**(b)** Which 3 edges would be added next to the MST in Kruskal's algorithm? Be sure to indicate the order in which they are added.

Answer: First (G,H), then (A,B), then (E,H).

**(c)** Which 3 edges would be added next to the BFS-tree by BFS? Be sure to indicate the order in which they are added.

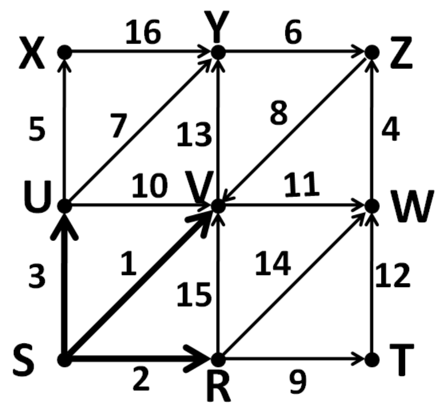
Answer: First (A,B), then (A,E), then (C,F).

**(d)** At this point in the running of Dijkstra's algorithm, $S$ has been taken off the top of the priority queue and marked as "visited". Which 4 vertices would be marked next in Dijkstra's algorithm, i.e. deleted from the priority queue and marked? What are the shortest paths, and their lengths, to these 4 vertices?

Answer: First D (shortest path S-D, length 1), then C (shortest path S-C, length 2), then A (shortest path S-A, length 3), then B (shortest path S-A-B, length 3+5=8).

**Question 1 (20 points).** We are running the following four algorithms on the graph below, where the algorithms have already "processed" the three bold-face edges:

- Dijkstra's algorithm for shortest paths, starting from $S$.
- Prim's algorithm for the Minimum Spanning Tree (MST), starting from $S$ (ignoring edge directions).
- Kruskal's algorithm for the Minimum Spanning Tree (MST) (ignoring edge directions).
- Breadth-First-Search (BFS) starting from $S$ (ignoring both edge directions and edge weights, but visiting neighboring vertices in lexicographic order).




**(a)** Which 3 edges would be added next to the MST in Prim's algorithm? Be sure to indicate the order in which they are added.

Answer: First (U,X), then (U,Y), then (Y,Z).

**(b)** Which 3 edges would be added next to the MST in Kruskal's algorithm? Be sure to indicate the order in which they are added.

Answer: First (W,Z), then (U,X), then (Y,Z).

**(c)** Which 3 edges would be added next to the BFS-tree by BFS? Be sure to indicate the order in which they are added.

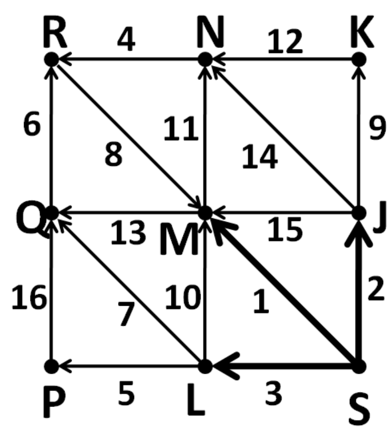
Answer: First (R,T), then (R,W), then (U,X).

**(d)** At this point in the running of Dijkstra's algorithm, $S$ has been taken off the top of the priority queue and marked as "visited". Which 4 vertices would be marked next in Dijkstra's algorithm, i.e. deleted from the priority queue and marked? What are the shortest paths, and their lengths, to these 4 vertices?

Answer: First V (shortest path S-V, length 1), then R (shortest path S-R, length 2), then U (shortest path S-U, length 3), then X (shortest path S-U-X, length 3+5=8).

**Question 1 (20 points).** We are running the following four algorithms on the graph below, where the algorithms have already "processed" the three bold-face edges:

- Dijkstra's algorithm for shortest paths, starting from $S$.
- Kruskal's algorithm for the Minimum Spanning Tree (MST) (ignoring edge directions).
- Prim's algorithm for the Minimum Spanning Tree (MST), starting from $S$ (ignoring edge directions).
- Breadth-First-Search (BFS) starting from $S$ (ignoring both edge directions and edge weights, but visiting neighboring vertices in lexicographic order).




**(a)** Which 3 edges would be added next to the MST in Prim's algorithm? Be sure to indicate the order in which they are added.

Answer: First (L,P), then (L,Q), then (Q,R).

**(b)** Which 3 edges would be added next to the MST in Kruskal's algorithm? Be sure to indicate the order in which they are added.

Answer: First (N,R), then (L,P), then (Q,R).

**(c)** Which 3 edges would be added next to the BFS-tree by BFS? Be sure to indicate the order in which they are added.

Answer: First (J,K), then (J,N), then (L,P).

**(d)** At this point in the running of Dijkstra's algorithm, $S$ has been taken off the top of the priority queue and marked as "visited". Which 4 vertices would be marked next in Dijkstra's algorithm, i.e. deleted from the priority queue and marked? What are the shortest paths, and their lengths, to these 4 vertices?

Answer: First M (shortest path S-M, length 1), then J (shortest path S-J, length 2), then L (shortest path S-L, length 3), then P (shortest path S-L-P, length 3+5=8).

**Question 2 (20 points).** In this problem, we will analyze a new algorithm for Minimum Spanning Tree (MST). It is based on the observation that for every vertex of a graph, the shortest edge incident on that vertex is part of an MST. In this problem, we assume we have an undirected connected graph G=(V, E), where edge weights can be positive or negative. (Part a is worth 6 points, and parts b-h are worth 2 points each.)

a) The shortest edge incident on any vertex is part of an MST. Prove this fact by filling in the blanks in the following proof by contradiction:

Assume for the sake of contradiction that the shortest edge $e_v$ (of weight $w_0$) incident on vertex $v$ is *not* part of any MST. Consider the union of an MST T and $e_v$, T $\bigcup \{e_v\}$, which will have a _____________ containing vertex $v$. This _____________ will have _____________ (*how many?*) edges incident on $v$. Assume we remove the edge $e$ of weight $w$ that was originally part of the MST T. Of course, $w$ is _____________ (*bigger than? less than? equal to?*) $w_0$ by the original assumption. Then, by removing $e$ from T $\bigcup \{e_v\}$, we obtain a new _____________ T' of weight _____________ (*at least? at most? equal to?*) the weight of T. Hence T' (containing edge $e_v$) is an MST - contradiction!

Answer: Assume for the sake of contradiction that the shortest edge $e_v$ (of weight $w_0$) incident on vertex $v$ is *not* part of any MST. Consider the union of an MST T and $e_v$, T $\bigcup \{e_v\}$, which will have a <u>cycle</u> containing vertex $v$. This <u>cycle</u> will have <u>two</u> (*how many?*) edges incident on $v$. Assume we remove the edge $e$ of weight $w$ that was originally part of the MST T. Of course, $w$ is <u>bigger than</u> (*bigger than? less than? equal to?*) $w_0$ by the original assumption. Then, by removing $e$ from T $\bigcup \{e_v\}$, we obtain a new <u>tree</u> T' of weight <u>at most</u> (*at least? at most? equal to?*) the weight of T. Hence T' (containing edge $e_v$) is an MST - contradiction!

The algorithm works by creating a series of graphs $F_i$ (i.e. $F_1, F_2, \ldots$). At each step $i$, we create graph $F_i$ from graph $F_{i-1}$ by contracting two nodes into one as shown below, and then updating the edges correspondingly, keeping only the shortest edge between any pair of vertices:

$F_0$ = G, set T is initially empty

$i = 1$

While $F_{i-1}$ has at least two vertices: (start step $i$)

  Initialize all vertices in $F_{i-1}$ to be unmarked

  For $k$=1, 2, ... up to the number of vertices in $F_{i-1}$

    If $v_k$ is unmarked contract $v_k$ and its nearest neighbor as follows:

      Find the shortest edge incident on $v_k$, call it $(v_k, v_l)$

      Mark $v_k$ and $v_l$

      Add $(v_k, v_l)$ to T

      Add a vertex $v_{k'}$ to graph $F_i$ ($v_{k'}$ is the "contracted" vertex in $F_i$ of both $v_k$ and $v_l$)

  For each edge $e = (v_a, v_b)$ of $F_{i-1}$, add an edge between the contracted vertices in $F_i$ (e.g. $v_{a'}$ and $v_{b'}$) under the conditions:

$v_{a'}$ and $v_{b'}$ are distinct

if there already is an edge between $v_{a'}$ and $v_{b'}$ in $F_i$, only keep the one of lower weight

$i = i + 1$

Return T

For the following questions, circle the correct answer (as we previously said, a step is performed every time we update from $F_{i-1}$ to $F_i$)

b) The total number of steps will be (circle the tightest bound)

$O(|E|log|E|)$ $\quad$ $O(\sqrt{|V|})$ $\quad$ $O(|E|)$ $\quad$ $O(log|V|)$

Answer: $O(log|V|)$

c) At each step, the amount of work is (circle the tightest bound)

$O(|E|)$ $\quad$ $O(|V|)$ $\quad$ $O(|E|^2)$ $\quad$ $O(|V|^2)$

Answer: $O(|E|)$

d) Hence, the running time of this algorithm, compared to Kruskal, seems to be assymptotically:

Faithful Faster $\quad$ The same $\quad$ Cannot be compared $\quad$ Slower

Answer: The same

By doing a more careful analysis, it turns out we can improve on part c. Let $|F_i|$ be the number of vertices of $F_i$.

e) How many edges can $F_i$ have in the worst case (circle the tightest bound)

$O(|F_i|^2)$ $\quad$ $O(|F_i|log|F_i|)$ $\quad$ $O(|F_i|)$ $\quad$ $O(|F_i|\sqrt{|F_i|})$

Answer: $O(|F_i|^2)$

f) Hence, the amount of work per step is the minimum of ____________ and ____________ (fill in the blanks)

Answer: Hence, the amount of work per step is the minimum of $\underline{O(|E|)}$ and $\underline{O(|F_i|^2)}$ (fill in the blanks)

Now find the total running time under the following assumptions:

g) If the graph is sparse ($|E| = \Theta(|V|)$) then the total running time is (circle the tightest bound)

$O(|E|\sqrt{|E|})$ $O(|E|log|E|)$ $O(|E|)$ $O(|E|^2)$

Answer: $O(|E|log|E|)$

h) If the graph is dense ($|E| = \Theta(|V|^2)$), then the total running time is (circle the tightest bound)

$O(|E|\sqrt{|E|})$ $O(|E|^2)$ $O(|E|log|E|)$ $O(|E|)$

Answer: $O(|E|)$

**Question 2 (20 points).** In this problem, we will analyze a new algorithm for Minimum Spanning Tree (MST). It is based on the observation that for every vertex of a graph, the shortest edge incident on that vertex is part of an MST. In this problem, we assume we have an undirected connected graph G=(V, E), where edge weights can be positive or negative. (Part a is worth 6 points, and parts b-h are worth 2 points each.)

a) The shortest edge incident on any vertex is part of an MST. Prove this fact by filling in the blanks in the following proof by contradiction:

Assume for the sake of contradiction that the shortest edge $e_v$ (of weight $w_0$) incident on vertex $v$ is *not* part of any MST. Consider the union of an MST T and $e_v$, T $\bigcup \{e_v\}$, which will have a ____________ containing vertex $v$. This ____________ will have ____________ (*how many?*) edges incident on $v$. Assume we remove the edge $e$ of weight $w$ that was originally part of the MST T. Of course, $w$ is ____________ (*bigger than? less than? equal to?*) $w_0$ by the original assumption. Then, by removing $e$ from T $\bigcup \{e_v\}$, we obtain a new ____________ T' of weight ____________ (*at least? at most? equal to?*) the weight of T. Hence T' (containing edge $e_v$) is an MST - contradiction!

Answer: Assume for the sake of contradiction that the shortest edge $e_v$ (of weight $w_0$) incident on vertex $v$ is *not* part of any MST. Consider the union of an MST T and $e_v$, T $\bigcup \{e_v\}$, which will have a <u>cycle</u> containing vertex $v$. This <u>cycle</u> will have <u>two</u> (*how many?*) edges incident on $v$. Assume we remove the edge $e$ of weight $w$ that was originally part of the MST T. Of course, $w$ is <u>bigger than</u> (*bigger than? less than? equal to?*) $w_0$ by the original assumption. Then, by removing $e$ from T $\bigcup \{e_v\}$, we obtain a new <u>tree</u> T' of weight <u>at most</u> (*at least? at most? equal to?*) the weight of T. Hence T' (containing edge $e_v$) is an MST - contradiction!

The algorithm works by creating a series of graphs $H_j$ (i.e. $H_1, H_2, \ldots$). At each step $j$, we create graph $H_j$ from graph $H_{j-1}$ by contracting two nodes into one as shown below, and then updating the edges correspondingly, keeping only the shortest edge between any pair of vertices:

$H_0$ = G, set T is initially empty

$j = 1$

While $H_{j-1}$ has at least two vertices: (start step $j$)

- Initialize all vertices in $H_{j-1}$ to be unmarked
- For $l$=1, 2, ... up to the number of vertices in $H_{j-1}$
  - If $u_l$ is unmarked contract $u_l$ and its nearest neighbor as follows:
    - Find the shortest edge incident on $u_l$, call it $(u_l, u_i)$
    - Mark $u_l$ and $u_i$
    - Add $(u_l, u_i)$ to T
    - Add a vertex $u_{l'}$ to graph $H_j$ ($u_{l'}$ is the "contracted" vertex in $H_j$ of both $u_l$ and $u_i$)
- For each edge $e = (u_b, u_c)$ of $H_{j-1}$, add an edge between the contracted vertices in $H_j$ (e.g. $u_{b'}$ and $u_{c'}$) under the conditions:

$u_{b'}$ and $u_{c'}$ are distinct

if there already is an edge between $u_{b'}$ and $u_{c'}$ in $H_j$, only keep the one of lower weight

$j = j + 1$

Return T

For the following questions, circle the correct answer (as we previously said, a step is performed every time we update from $H_{j-1}$ to $H_j$)

b) At each step, the amount of work is (circle the tightest bound)

$O(|V|)$ $\quad$ $O(|E|)$ $\quad$ $O(|E|^2)$ $\quad$ $O(|V|^2)$

Answer: $O(|E|)$

c) The total number of steps will be (circle the tightest bound)

$O(|E| log|E|)$ $\quad$ $O(log|V|)$ $\quad$ $O(\sqrt{|V|})$ $\quad$ $O(|E|)$

Answer: $O(log|V|)$

d) Hence, the running time of this algorithm, compared to Kruskal, seems to be assymptotically:

Faithful Faster $\quad$ Cannot be compared $\quad$ The same $\quad$ Slower

Answer: The same

By doing a more careful analysis, it turns out we can improve on part c. Let $|H_j|$ be the number of vertices of $H_j$.

e) How many edges can $H_j$ have in the worst case (circle the tightest bound)

$O(|H_j| log|H_j|)$ $\quad$ $O(|H_j|^2)$ $\quad$ $O(|H_j|)$ $\quad$ $O(|H_j|\sqrt{|H_j|})$

Answer: $O(|H_j|^2)$

f) Hence, the amount of work per step is the minimum of ______ and ______ (fill in the blanks)

Answer: Hence, the amount of work per step is the minimum of $\underline{O(|E|)}$ and $\underline{O(|H_j|^2)}$ (fill in the blanks)

Now find the total running time under the following assumptions:

g) If the graph is sparse ($|E| = \Theta(|V|)$) then the total running time is (circle the tightest bound)

$O(|E|\sqrt{|E|})$ $\quad$ $O(|E|)$ $\quad$ $O(|E|^2)$ $\quad$ $O(|E|log|E|)$

Answer: $O(|E|log|E|)$

h) If the graph is dense ($|E| = \Theta(|V|^2)$), then the total running time is (circle the tightest bound)

$O(|E|\sqrt{|E|})$ $\quad$ $O(|E|)$ $\quad$ $O(|E|^2)$ $\quad$ $O(|E|log|E|)$

Answer: $O(|E|)$

**Question 2 (20 points).** In this problem, we will analyze a new algorithm for Minimum Spanning Tree (MST). It is based on the observation that for every vertex of a graph, the shortest edge incident on that vertex is part of an MST. In this problem, we assume we have an undirected connected graph G=(V, E), where edge weights can be positive or negative. (Part a is worth 6 points, and parts b-h are worth 2 points each.)

a) The shortest edge incident on any vertex is part of an MST. Prove this fact by filling in the blanks in the following proof by contradiction:

   Assume for the sake of contradiction that the shortest edge $e_v$ (of weight $w_0$) incident on vertex $v$ is *not* part of any MST. Consider the union of an MST T and $e_v$, T $\bigcup$ $\{e_v\}$, which will have a _____________ containing vertex $v$. This _____________ will have _____________ (*how many?*) edges incident on $v$. Assume we remove the edge $e$ of weight $w$ that was originally part of the MST T. Of course, $w$ is _____________ (*bigger than? less than? equal to?*) $w_0$ by the original assumption. Then, by removing $e$ from T $\bigcup$ $\{e_v\}$, we obtain a new _____________ T' of weight _____________ (*at least? at most? equal to?*) the weight of T. Hence T' (containing edge $e_v$) is an MST - contradiction!

   Answer: Assume for the sake of contradiction that the shortest edge $e_v$ (of weight $w_0$) incident on vertex $v$ is *not* part of any MST. Consider the union of an MST T and $e_v$, T $\bigcup$ $\{e_v\}$, which will have a <u>cycle</u> containing vertex $v$. This <u>cycle</u> will have <u>two</u> (*how many?*) edges incident on $v$. Assume we remove the edge $e$ of weight $w$ that was originally part of the MST T. Of course, $w$ is <u>bigger than</u> (*bigger than? less than? equal to?*) $w_0$ by the original assumption. Then, by removing $e$ from T $\bigcup$ $\{e_v\}$, we obtain a new <u>tree</u> T' of weight <u>at most</u> (*at least? at most? equal to?*) the weight of T. Hence T' (containing edge $e_v$) is an MST - contradiction!

The algorithm works by creating a series of graphs $L_k$ (i.e. $L_1, L_2, \ldots$). At each step $k$, we create graph $L_k$ from graph $L_{k-1}$ by contracting two nodes into one as shown below, and then updating the edges correspondingly, keeping only the shortest edge between any pair of vertices:

$L_0 =$ G, set T is initially empty

$k = 1$

While $L_{k-1}$ has at least two vertices: (start step $k$)

- Initialize all vertices in $L_{k-1}$ to be unmarked
- For $i$=1, 2, ... up to the number of vertices in $L_{k-1}$
  - If $w_i$ is unmarked contract $w_i$ and its nearest neighbor as follows:
    - Find the shortest edge incident on $w_i$, call it $(w_i, w_j)$
    - Mark $w_i$ and $w_j$
    - Add $(w_i, w_j)$ to T
    - Add a vertex $w_{i'}$ to graph $L_k$ ($w_{i'}$ is the "contracted" vertex in $L_k$ of both $w_i$ and $w_j$)
- For each edge $e = (w_c, w_d)$ of $L_{k-1}$, add an edge between the contracted vertices in $L_k$ (e.g. $w_{c'}$ and $w_{d'}$) under the conditions:
  - $w_{c'}$ and $w_{d'}$ are distinct

if there already is an edge between $w_{c'}$ and $w_{d'}$ in $L_k$, only keep the one of lower weight

$k = k + 1$

Return T

For the following questions, circle the correct answer (as we previously said, a step is performed every time we update from $L_{k-1}$ to $L_k$)

b) At each step, the amount of work is (circle the tightest bound)

$O(|V|)$ $O(|E|^2)$ $O(|E|)$ $O(|V|^2)$

Answer: $O(|E|)$

c) The total number of steps will be (circle the tightest bound)

$O(|E|log|E|)$ $O(\sqrt{|V|})$ $O(log|V|)$ $O(|E|)$

Answer: $O(log|V|)$

d) Hence, the running time of this algorithm, compared to Kruskal, seems to be assymptotically:

Faster Cannot be compared Slower The same

Answer: The same

By doing a more careful analysis, it turns out we can improve on part c. Let $|L_k|$ be the number of vertices of $L_k$.

e) How many edges can $L_k$ have in the worst case (circle the tightest bound)

$O(|L_k|log|L_k|)$ $O(|L_k|)$ $O(|L_k|^2)$ $O(|L_k|\sqrt{|L_k|})$

Answer: $O(|L_k|^2)$

f) Hence, the amount of work per step is the minimum of ______ and ______ (fill in the blanks)

Answer: Hence, the amount of work per step is the minimum of $\underline{O(|E|)}$ and $\underline{O(|L_k|^2)}$ (fill in the blanks)

Now find the total running time under the following assumptions:

g) If the graph is dense ($|E| = \Theta(|V|^2)$), then the total running time is (circle the tightest bound)

$O(|E|)$ $\quad$ $O(|E|\sqrt{|E|})$ $\quad$ $O(|E|^2)$ $\quad$ $O(|E|log|E|)$

Answer: $O(|E|)$

h) If the graph is sparse ($|E| = \Theta(|V|)$) then the total running time is (circle the tightest bound)

$O(|E|log|E|)$ $\quad$ $O(|E|\sqrt{|E|})$ $\quad$ $O(|E|)$ $\quad$ $O(|E|^2)$

Answer: $O(|E|log|E|)$

**Question 3 (20 points)**

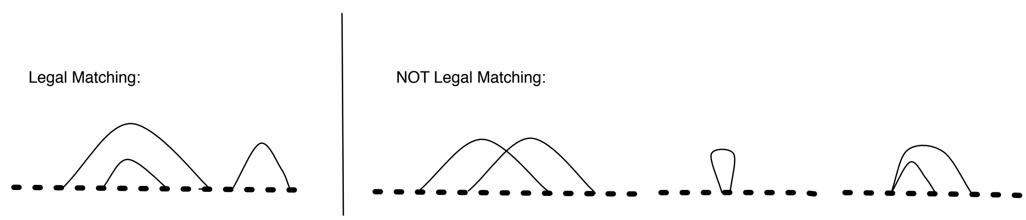
*Only answers for version 1 of the question is supplied, other versions differ in variable names only.*

In this question we will be given a string of characters, $s = (s_1, s_2, \ldots, s_n)$.

A set $D$ is called a "legal set of matchings" if its elements are pairs of indices $(a, b)$ where $1 \le a < b \le n$ and $s_a = s_b$, and any two elements $(a, b)$ and $(x, y)$ are either "disjoint", i.e $a < b < x < y$ or $x < y < a < b$, or "nested", i.e. $a < x < y < b$ or $x < a < b < y$.

Notice that if two pairs $(a, b)$ and $(x, y)$ are neither disjoint or nested, then either they share an end point $a = x$ or $a = y$ or $b = x$ or $b = y$, or they "partially overlap", $a < x < b < y$ or $x < a < y < b$. In a legal set of matchings $D$, no 2 elements may share an endpoint or partially overlap.

Graphically, if we write $s$ on a linear line, and draw every match we pick $(a, b)$ as an edge between $s_a$ and $s_b$ (the edge is drawn above the string, see figure), a set is a legal set of matchings if the edges representing the matchings don't intersect.




We want to find an efficient dynamic programming algorithm that returns the size of the maximal legal set of matchings for a given string $s$.

a. (4 points) Complete the following subproblem definition:

$K(a, b)$ is the size of the maximal legal set of matchings ________________________________

Answer:

$K(a, b)$ is the size of the maximal legal set of matchings ___ in the substring of $s$ from index $a$ to index $b$.

b. (6 points) We define:

$$IsMatch(a,b) = \begin{cases} 1 & \text{if } s_a = s_b \\ 0 & \text{if } s_a \neq s_b \end{cases}$$

Write out the the computation for $K(a,b)$, using previous subproblems and the $IsMatch$ function:
Hint: It is helpful to consider the two distinct cases of trying to match $a$ with $b$, or not trying to do so.

$K(a,b) = \max \{$ ________________________________ $\}$

Answer:

$$K(a,b) = \max \; \{IsMatch(a,b) + K(a+1, b-1), \; \max_{a \leq x < b}\{K(a,x) + K(x+1,b)\}\}$$

c. (5 points) Write the pseudocode for your algorithm. The base case is $\forall a \quad K(a,a) = 0$

Answer:
Iterations:

**for** $j = 0, \ldots, n-1$ **do**
  **for** $a = 1, \ldots, n$ **do**
    **if** $a + j \leq n$ **then**
      compute $K(a, a+j)$
    **end if**
  **end for**
**end for**
return $K(1,n)$

d. (5 points) What is the running time of your algorithm?

Answer: $O(n^3)$

Short Explanation:

Answer: Every calculation of $K(a,b)$ takes $O(n)$ and there are two nested loops, each taking $O(n)$.