Algorithms. A Collection of Problems

July 9, 2012

★ **Trivial.**

★★ **Easy. Directly application of the textbook result.**

★★★ **Medium. One observation or one easy trick should do the work.**

★★★★ **Difficult. May require more than one tricks.**

★★★★★ **Difficult. May require more than one clever tricks.**

1. ★ (Longest path in a DAG) We are given a directed acyclic graph G and two specific vertices s and t in G. Each edge in the graph has a length. Design a polynomial time algorithm that finds the longest path from s to t. (if there is no path from s to t, your algorithm should be able to detect the fact.)

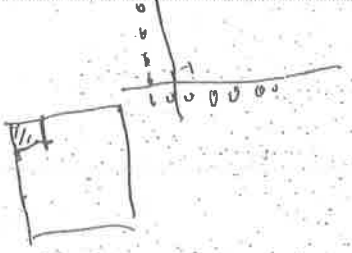
2. ★★ (Finding the maximum area polygon) We are given a unit circle and n points on the circle. Design a polynomial time algorithm that, given a number $m \leq n$, finds m points (out of n points) such that the area of the polygon formed by the m points is maximized.

3. ★ (Longest palindrome subsequence) A palindrome is a nonempty string over some alphabet that reads the same forward and backward. For example, $aaaabaaaa$, 00000, $abcddcba$ are all palindrome. Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. Your algorithm should run in time better than $O(n^3)$.

4. ★★ (Matrix-chain multiplication) We state the matrix-chain multiplication problem as follows: given a chain of matrices $A_1 A_2 \ldots A_n$ where A_i has dimension $p_{i-1} \times p_i$. (Note that the number of columns of A_i must be the same as the number of rows of A_{i+1}.) Assume multiplying a $x \times y$ matrix with a $y \times z$ matrix takes xyz scalar multiplications. Design a polynomial time to find a fully parenthesization of the product $A_1 \ldots A_n$ in a way that minimizes the number of scalar multiplications.

   For example, consider $A_1 A_2 A_3$ of three matrices with the dimensions of $10 \times 100, 100 \times 5, and 5 \times 50$, respectively. If we multiply according to the parenthesization $((A_1 A_2) A_3)$, we perform $5000 + 2500 = 7500$ scalar multiplications. However, if instead we multiply according to the $A_1(A_2 A_3)$, we perform $25,000 + 50,000 = 75,000$ scalar multiplications. Clearly, the first one is much better.

5. ★★ (Viterbi algorithm) We can use dynamic programming on a directed graph $G(V; E)$ for speech recognition. Each edge $(u, v) \in E$ is labeled with a sound from a finite set Σ of sounds. The labeled graph is a formal model of a person speaking a restricted language. Each path in the graph starting from a distinguished vertex $v_0 \in V$ corresponds to a possible sequence of sounds produced by the model. We define the label of a directed path to be the concatenation of the labels of the edges on that path.



(a) Describe a polynomial time algorithm that, given an edge-labeled graph G with distinguished vertex v_0 and a sequence $s = \langle \sigma_1, \ldots, \sigma_k \rangle$ of sounds from Σ, returns a path in G that begins at v_0 and has s as its label s, if any such path exists. Otherwise, the algorithm should return NO-SUCH-PATH.

(b) Now, suppose that every edge (u, v) has an associated nonnegative probability $p(u, v)$ of traversing the edge. and thus producing the corresponding sound. The sum of the probabilities of the edges leaving any vertex equals 1. The probability of a path is defined to be the product of the probabilities of its edges. We can view the probability of a path beginning at v_0 as the probability that a random walk beginning at v_0 will follow the specified path, where we randomly choose which edge to take leaving a vertex u according to the probabilities of the available edges leaving u. Extend your answer to part (a) so that if a path is returned, it is a most probable path starting at v_0 and having label s. Analyze the running time of your algorithm.

6. ★★★ (Edit distance) In order to transform one string x to a target string y, we can perform various edit operations. Our goal is, given x and y, to produce a series of edits that change x to y. We may choose from among edit operations:

(a) Insert a letter, (e.g., changing 100 to 1001 takes one insertion)

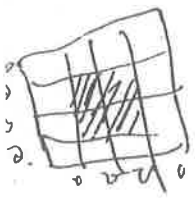
(b) Delete a letter, (e.g., changing 100 to 10 takes one deletion)

(c) Replace a letter by another (e.g., you need do one replacement to change 100 to 000).

Design an efficient algorithm that finds a series of edit operations that change x to y and the total number of edits is minimized.

7. ★★★★ (Four Russians Speedup) In this problem, we explore an interesting trick to speed up a dynamic program. We use the edit distance problem as an example (See the last problem). We assume the size of the alphabet is constant. Suppose the two strings are S_1 and S_2, both of length n. To start with, your solution for the last problem must run in $O(n^2)$ time. If it is so, in the dynamic program for the problem, you need to fill a two dimensional table M. In fact, we can fill out this table in a more clever way such that the running time can be improved to $O(n^2/\log n)$ (I know, it is a small improvement and of little practical interests for this particular problem. But the same trick has been used somewhere else to make a huge difference). In this trick, we need to a bit preprocessing to make many small tables. Then we fill only a subset of entries of the dynamic program table. The value of a table entry we are filling depends on the values of some entries we have already filled and the list we made in the beginning.

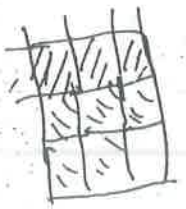
To make is a little bit more formal. We define a t-block to be a $t \times t$ squares in the dynamic programming table. Let $t = \frac{\log n}{4}$. We first observe that the distance values in a t-block starting in position (i, j) are a function of the values of its first row and first colum and substrings $S_1[i, \ldots, i + t - 1]$ and $S_2[i, \ldots, i + t - 1]$.



Now, let us observe another interesting fact: In any row or column, the values of two adjacent cells differ by at most 1.

- Please prove this fact.

We say two t-blocks B_1, B_2 are offset-equivalent if there is a number c, such that $B_1[i, j] = B_2[i, j] + c$ for all i, j.



- There are C types of t-blocks (up to offset-equivalence). Show how large C is.

It is obvious we can fill a t-block in $O(t^2)$ times. Therefore, filling all C types of blocks takes $O(Ct^2)$ times. These are the small tables we produce for preprocessing. How, we start to fill up the dynamic table M. We are going to fill up only $O(n^2/\log n)$ entries.

- We have given you enough hints. Now, it is up to you to develop the entire algorithm which should run in $O(n^2/\log n)$ time. In particular, you need to describe which entries of M that need to be filled and how to compute their values.

8. ★★ Solve the following two recurrences:

(a) $T(n) = 2T(n/2) + n\log n$.

(b) $T(n) = 2T(\sqrt{n}) + \log n$.

9. ★★★ **(Maximal Common Subsequence)** String C is a subsequence of string A if C can be obtained by deleting some letters from A. For example, *ade* is a substring of *abcde*. We are given two string $A = a_1a_2\ldots a_m$ and $B = b_1b_2\ldots b_n$. Design an algorithm that finds a common subsequence of A and B using $O(mn)$ time and $O(m+n)$ space. (You will get half of the points if you can find a polynomial time algorithm, but the running time (and/or) the space are worse that the stated.)

10. ★ (Stick Game) There is a pile of n sticks. Two players A and B take turns removing 1 or 4 sticks. A starts first. The players who removes the last stick wins. Design a polynomial time algorithm (polynomial in n) that decides which player has a winning strategy (i.e., no matter how the opponent plays, the player can take certain moves to win the game.)

11. ★★★ (Monge Matrix) An $m \times n$ array A of real number is a *Monge matrix* if

$$A[i,j] + A[k,l] \le A[i,l] + A[k,j], \forall i < k \text{ and } j < l. \quad \cdots (1)$$

Answer the following questions. Let $f(i)$ the column index of the leftmost minimum element of row i. For example:

$$\begin{pmatrix} 10 & 9 & 12 & 10 \\ 11 & 10 & 13 & 10 \\ 9 & 8 & 10 & 7 \\ 11 & 10 & 11 & 8 \end{pmatrix}$$

is a Monge Matrix. $f(1) = 2, f(2) = 2, f(3) = 4, f(4) = 4$.

(a) Show that $f(1) \le f(2) \le \ldots \le f(m)$.

(b) Given an $m \times n$ Monge matrix, Design an algorithm that computes $f(1), \ldots, f(m)$ in $O(m + n\log m)$ time.

Just a side note: In fact, there is an $O(m+n)$ time algorithm to compute the row minima for Monge Matrix. It is called the SMAWK algorithm (The initials of five authors).

12. ★★ (Unit tasks scheduling) We are given a set of unit-time tasks. Each task i is supposed to finish by time d_i (d_i is an integer). Each task i is also associated with a penalty w_i if i is not finished by time d_i. There is no penalty if we finish task i by its deadline. Design a polynomial time that find a schedule that minimizes the total penalty.

13. ★★ (Coin changing) Suppose that the coins are in the denominations that are powers of c, i.e., $c^0, c^1, c^2, \ldots, c^k$, for some integers $c > 1, k \geq 1$. You are asked to change for n cents using the fewest number of coins. Show that the greedy algorithm yields an optimal solution. (The greedy algorithm first tries the coin with the largest denomination , then the coin with the second largest denomination, and so on).

14. ★★ (Schedule to minimize completion time) You are given a set $S = \{a_1, a_2, \ldots, a_n\}$ tasks. Task a_i is released at time r_i (you can not start a_i before r_i) and requires p_i units of time to process. You machine can process one task at each time. Assume *preemption* is allowed, so that you can suspend a task and resume it at a later time. For a particular schedule S, we denote the completion time of task a_i (the time when a_i is finished) to be $c_S(a_i)$ Give a polynomial time algorithm that finds a schedule S of all tasks and minimizes the total completion time $\sum_{i=1}^{n} c_S(a_i)$.

15. ★★★ We are given an unweighted undirected graph G. Let M be a matching in G that has no augmenting path of length smaller than $2t + 1$. Let M^* be the maximum matching in G. Show that $|M| \geq \frac{t}{t+1}|M^*|$.

16. ★★★ We are given n jobs and one server. Each job j is associated with a profit p_j, a release time r_j and completion time c_j. If we decide to schedule job j, the server has to process it continuously from time r_j to c_j and we can get a profit p_j. No partial profit can be obtained if the job is not finished. The server can process at most k jobs at any time. Design a polynomial time algorithm that finds a feasible schedule such that the total profit we can get is maximized.

17. ★★★ Given an undirected graph $G(V, E)$, a feedback set is a set $X \subseteq V$ with the property that $G - X$ has no cycles. The undirected feedback set problem asks whether G contains a feedback set of size at most k. Show that the problem is NP-complete.

18. ★★ (Rearrangeable Matrix) Let $M = \{m_{ij}\}_{1 \leq i,j \leq n}$ be an $n \times n$ 0/1 matrix. We say M is rearrangeable if there exist a permutation π of rows and a permutation τ of columns such that the matrix $M' = \{m'_{ij}\}_{1 \leq i,j \leq n}$ obtained by permuting the rows of M according to π and permuting the columns of M according to τ has all its diagonal entries (i.e., m'_{ii} for $1 \leq i \leq n$) equal to 1.

(a) Give an example of a matrix M that is not rearrangeable, but for which at least one entry in each row and each column is equal to 1.(3 pts)

(b) Give a polynomial time algorithm that determines whether a matrix is rearrangeable. (7 pts)

19. ★★★ (Turán's bound) We are given a graph $G(V, E)$. The degree of a vertex v, denoted by $deg(v)$, is defined to be the number of edges incident on v. A subset S of vertices is called *an independent set* if there is no edge connecting any two vertices in S. Let $\alpha(G)$ be the size (i.e., number of vertices) of the maximum independent set in graph G. Prove the following inequality:

$$\alpha(G) \geq \sum_{v \in V} \frac{1}{deg(v) + 1}.$$

(Hint: Consider the greedy algorithm which repeatedly adds the vertex with lowest degree and delete all its neighbors.)

20. ★★★ (Multiple Interval Scheduling) In this problem, we have a single processor which can only work on one job at any single point in time. We are also given a set of jobs, each of them requiring a

set of disjoint intervals of time during which it needs to be processed on the processor. For example, a job could require the processor form 10am to 11am and 1pm to 2pm. We would like to answer the following question: For a set of jobs and a given number k, is it possible to process at least k jobs in the processor. Show that the problem is NP-complete. (Hint: Maximum Independent Set is a possible candidate for the reduction.)

21. ★★★ You have m slow machines and k fast machines. You are given a set of n jobs. Job i takes time t_i to process on a slow machine and time $t_i/2$ to process on a fast machine. Assign each job to a machine such that the makespan is minimized. (The makespan is the maximum load of any machine.) Design a polynomial time approximation algorithm that produces an assignment with a makespan at most 3 times the optimum.

22. ★★ (Densest k-Subgraph) We are given an undirected graph $G(V, E)$. The densest k-subgraph problem asks for a subset S of k vertices such that the number of edges in the induced graph $G[S]$ is maximized. Formulate the decision version of the problem and show it is NP-Complete.

23. ★★★ (maximum coverage) We are given a ground set U of n element and a collection of subsets $S_1, \ldots, S_m$. The goal is to choose k subsets and maximize the cardinality of their union.

(a) Formulate the decision version of the problem and show it is NP-Complete.

(b) Design a polynomial time approximation algorithm for the problem. Your algorithm should have an approximation ratio of $1 - \frac{1}{e}$ in order to get full credit.

24. ★★★ (Polynomial Multiplication) You are given $2n$ numbers $a_1, b_1, \ldots, a_n, b_n$. Consider the polynomial $P(x) = \prod_{i=1}^{n}(a_i x + b_i)$. Design an algorithm that computes the expansion of $P(x)$. In order to get full credit, your algorithm should run in time $o(n^2)$ (Note: small o).

25. ★★ (Longest increasing subsequence) You are given a sequence S of n numbers $a_1, a_2, \ldots, a_n$. A subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ is increasing if $i_1 < i_2 < \ldots < i_k$ and $a_{i_1} < a_{i_2}, \ldots < a_{i_k}$. For example, $S = \{2, 10, 7, 11, 8, 6, 12\}$ is the given sequence and $\{2, 10, 11, 12\}$ is an increasing subsequence. Design a polynomial time algorithm that finds an longest increasing subsequence in the given sequence S.

26. There is a round table divided into 4 equal quadrants, with one cup in each quadrant. The quadrants are labeled with letters (A, B, C, D) that do not move. Initially, each cup is randomly face-up or face-down. You are blindfolded and put in front of the table. On each turn of the game, you instruct a genie to flip the cups in whichever positions you choose (e.g., you may say "flip the cups in A and B"), possibly choosing no cups or possibly all four. The genie complies. At this point, if all the cups on the table are face-up, the genie will tell you that you have won the game and are free to go. If not, he rotates the cups randomly (possibly not rotating them) and you play another turn.

(a) ★★ Give a strategy to win this game in a finite number of moves (the solution is not unique). Remark: the outcome of "rotation" the four cups is one of the four possible positions: the cups originally at (A,B,C,D) can be at (A,B,C,D), (B,C,D,A), (C,D,A,B), or (D,A,B,C). It is not an arbitrary permutation.

(b) ★★★★ Show that there is a strategy that guarantees a win if the number of cups N is a power of 2.

(c) ★★★★ Show that there is no strategy that guarantees a win if the number of cups N is not a power of 2. (You can get partial credits if you can give a proof for the case where N is odd and larger than 3.)

27. ★★★★★ This problem is a generalization of the above problem. There are fixed positive integers N and K, each at least 2. We have an unknown vector V of length N of integers between 0 and $K-1$. During each move, we propose another vector W of length N, of integers between 0 and $K-1$. A genie selects a secret integer J between 0 and $N-1$ and rotates W by J positions: $\text{Rot}(W, J)$ is the last $N-J$ entries of W, followed by the first J entries of W. The genie then replaces V by $(V + \text{Rot}(W, J)) (\text{mod } K)$. If the new value of V is all 0, we win; otherwise we continue to play. For what pairs of integers (N, K) is there a fixed finite sequence of moves (a sequence of vectors W) such that if we apply this sequence in order, we are guaranteed to win by the time the sequence is finished? Prove that these pairs admit such a sequence, and that no other pairs do.

28. ★★★ In this problem, we are going to work with a very popular class of objects in computer science - graphs (图). A graph consists a set V of vertices (点) and a set E of edges (边). An edge connects two vertices. The degree of a vertex v, denoted by $deg(v)$, is defined to be the number of edges incident on v. Now, we consider a very classic problem in computer science - the traveling salesman problem. A path is a set of edges of the form $\{(v_1, v_2), (v_2, v_3), (v_3, v_4), \ldots, (v_{k-1}, v_k)\}$ and a cycle is a path whose head and tail are the same vertex. A Hamitonian cycle is a cycle that goes through all vertices exactly once. See Figure 2 for an example. Prove the following statement: Every graph with $n \geq 3$ vertices and minimum degree at least $\lceil n/2 \rceil$ has a Hamitonian cycle. (Hint: consider the longest path.)

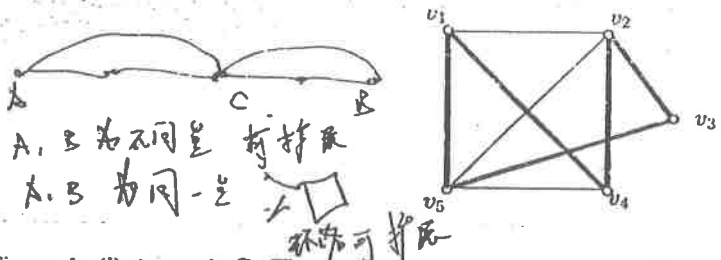
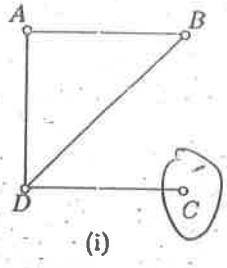


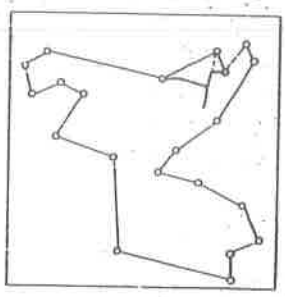
Figure 1: (i) A graph G. The vertices are v_1, v_2, v_3, v_4, v_5. There are 8 edges. The degree of v_1 is 3 and the degree of v_2 is 4. $\{(v_1, v_4), (v_4, v_2), (v_2, v_3), (v_3, v_5), (v_5, v_1)\}$ is Hamitonian cycle (indicated by the thicker edges).

29. ★★ Catch a car without knowing anything. We have an infinite road (i.e., $(-\infty, +\infty)$) and a car whose initial position p (at time 0) is some unknown integer. The car is running at a fixed velocity v. v is some unknown but fixed integer. Note that v can be negative which means the car is heading towards $-\infty$. At each time step, you can make at most one query of the following form: Is the car at position x now? (you can choose the integer x). Design a query strategy such that you can guarantee you will get a "yes" answer in finite time. (HINT: It is easy to see that the position of the car at time t is $p + vt$.)

30. ★★★ Turán's bound. In this problem, we are going to work with a very popular class of objects in computer science - graphs (图). A graph consists a set V of vertices (点) and a set E of edges (边). An edge connects two vertices. The degree of a vertex v, denoted by $deg(v)$, is defined to be



(i)



(ii)

Figure 2: (i) A graph G. The vertices are A, B, C, D. The edges are AB, AD, BD, DC. The degree of A is 2 and the degree of D is 3. The set {A,C} is an independent set of size two, which also happens to be the maximum one. So, $\alpha(G) = 2$. But $\{A, B, C\}$ is not an independent set, due to the existence of edge AB. (ii) A traveling salesman tour.

the number of edges incident on v. See Figure 2(i) for an example. A subset S of vertices is called *an independent set (独立集)* if there is no edge connecting any two vertices in S. Let $\alpha(G)$ be the size (i.e., number of vertices) of the maximum independent set in graph G. Prove the following inequality:

$$\alpha(G) \geq \sum_{v \in V} \frac{1}{deg(v) + 1}.$$

31. ★★★★ Traveling salesman in the unit square. In this problem, we are considering a very classic problem in computer science - the traveling salesman problem. We are given a set of n points in the unit square ($[0, 1] \times [0, 1]$). For two points v, u, we use $|uv|$ to denote the Euclidean distance between u and v. A traveling salesman tour is a tour that starts at some starting point, goes through all points exactly once and ends at the starting point. See Figure 2(ii) for an example.

a. Prove that there is a tour $\{v_1, v_2, \ldots, v_n, v_1\}$ (i.e., the tour visits v_1 first, then v_2 and so on) such that

$$|v_1v_2|^2 + |v_2v_3|^2 + \ldots + |v_{n-1}v_n|^2 + |v_nv_1|^2 \leq 4$$

b. Prove that there is a tour $\{v_1, v_2, \ldots, v_n, v_1\}$ such that

$$|v_1v_2| + |v_2v_3| + \ldots + |v_{n-1}v_n| + |v_nv_1| \leq 2\sqrt{n}$$

32. (a) ★ Show that a graph with maximum degree Δ can be colored in $\Delta + 1$ colors.

(b) ★ We are given a 3-colorable graph $G(V, E)$. In other words, it is possible to color the vertices of G using 3 colors such that no edge is incident on two vertices with the same color. Show that 2-coloring the neighbors of any vertex (if possible) can be done in polynomial time.

(c) ★★★★ We are given a 3-colorable graph $G(V, E)$. Design a polynomial time algorithm that finds an $O(\sqrt{|V|})$-coloring of G.

33. ★★★★★ You have a graph G. Each edge e has two positive costs A_e and B_e. Find a spanning tree T that minimizes $(\sum_{e \in T} A_e) \times (\sum_{e \in T} B_e)$. You algorithm should run in polynomial time.

34. ★★★ Initially, we have n empty bins. In each round, we throw a ball into a uniformly random chosen bin. Let T be number of rounds needed such that no bin is empty. Show that $\mathbb{E}[T] = nH_n$ where $H_n = \sum_{i=1}^{n} \frac{1}{n}$.

35. ★★★ Show that finding a min-cost matching with exactly k edges in a bipartite graph can be solved in polynomial time.

36. ★★★ A vertex cover is a set of vertices such that each edge is incident on at least one vertex in the set. An independent set is a set of vertices such that no edge joins two vertices in the set (i.e., no two vertices in the set are adjacent.) Prove the following statements:

(a) In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover.

(b) In any bipartite graph, the number of vertices in a maximum independent set equals the total number of vertices minus the number of edges in a maximum matching.

37. ★★★★ Given an undirected edge weighted graph $G(V, E)$, design a polynomial time algorithm that finds a minimum weight subset E' of edges such that each vertex is incident on at least one edges in E'.

38. ★★★★ A cycle cover of an undirected graph is a collection of vertex-disjoint cycles such that each vertex belongs to exactly one cycle. We require each cycle contains at least three vertices. In an edge weighted graph, the weight of a cycle cover is the sum of the weights of all its edges. Given an undirected edge weighted graph $G(V, E)$, design a polynomial time algorithm that finds a cycle cover with the minimum total weight.