ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2021

Problem Set 4

Due: 23:59, May 18, 2021

1. Submit your solutions to Gradescope (www.gradescope.com).
2. In "Account Settings" of Gradescope, set your FULL NAME to your Chinese name and enter your STUDENT ID correctly.
3. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
4. When submitting your homework, match each of your solution to the corresponding problem number.

Problem 1:

The complexity class EXPTIME is defined to be the set of all decision problems which can be solved in exponential time, i.e. in $O(2^{n^k})$ time, for some fixed constant $k > 0$ on input of size n. In this problem, you will show PSPACE $\subseteq$ EXPTIME. To do this, first read about the *Turing machine* model of computation on Wikipedia or Baidu Baike. Note that PSPACE is the set of all problems solvable by a Turing machine using a polynomial size tape, while EXPTIME is the set of problems solvable by a Turing machine running in exponential time.

(a) Consider an algorithm which runs on a Turing machine whose tape has only one cell. That is, the algorithm is allowed to read or write to a single tape location, but cannot move its head. Show that such an algorithm must terminate in $O(1)$ time steps, or otherwise will loop forever.

(b) The previous problem established a relationship between an algorithm with space complexity 1 and its maximum running time. Extend the proof to show PSPACE $\subseteq$ EXPTIME. That is, show that an algorithm using polynomial space will either terminate in an exponential number of steps, or loop forever.

Problem 2:

Suppose that for some decision problem, we have an algorithm which on any instance computes the correct answer with probability at least $2/3$. We wish to reduce the probability of error by running the algorithm n times on the same input using independent randomness between trials and taking the most common result. Using Chernoff bounds, give an upper bound on the probability that this new algorithm produces an incorrect result.

Problem 3:

An *Erdos-Renyi random graph* $G(n, p)$ is constructed as follows: initially, there are $n \geq 2$ nodes and no edges. We iterate through all pairs of nodes, and for each pair we add an undirected edge between the nodes with probability p. Let V be the set of n nodes, and E be the set of edges created by this process. Then $G(n, p) = (V, E)$.

(a) Suppose we set $p = \frac{18 \log n}{n}$. Prove that with probability at least $1 - \frac{1}{n^2}$, the maximum degree for any node in $G(n, p)$ is $O(\log n)$.

(b) Suppose now $p = \frac{1}{2n}$. A *triangle* is defined as three distinct nodes $u, v, w \in V$, such that the edges $(u, v), (v, w)$ and (u, w) are all in E. Prove that with probability at least $\frac{7}{8}$, $G(n, p)$ does not contain any triangles.

Problem 4:

A *cut* in a graph is a partition of the vertices into two parts A and B. The size of the cut is the number of edges with one end in A and the other in B. A *minimum cut* is a cut of minimum size. Karger's contraction algorithm is a well-known randomized algorithm to compute a minimum cut of a connected graph. An excellent explanation of the algorithm is provided at `https://www.wikiwand.com/en/Karger%27s_algorithm`. The pseudocode for this algorithm is shown in Algorithm 1.

Algorithm 1. Contraction algorithm

```
1: procedure contract($G = (V, E)$):
2: while $|V| > 2$ do
3:     choose $e \in E$ uniformly at random
4:     $G \leftarrow G/e$
5: return the only cut in G
end
```

By modifying Karger's algorithm, give a Monte Carlo algorithm to find the *second smallest* cut in the graph with high probability.

Problem 5:

Suppose that you want to estimate the fraction f of people who drink Coke in Shanghai. Assume that you can select a Shanghai resident uniformly at random and determine whether they drink Coke. Also, assume you know a lower bound $0 < a < f$. Design a procedure for estimating f by some $\hat{f}$ such that $Pr[|f - \hat{f}| > \epsilon f] < \delta$, for any choice of constants $0 < a, \epsilon, \delta < 1$. What is the smallest number of residents you must query?

Problem 6:

Graph partitioning is the problem of dividing a graph into multiple pieces while optimizing for certain objectives. In particular, the *balanced cut* problem seeks to divide a graph with $2n$ nodes into two pieces A and B, each with n nodes, such that the total weight of all edges crossing A and B is minimized. This problem has a number of applications, for example in parallel computing and computer vision. Balanced cut is NP-complete. The Kernighan–Lin (KL) algorithm is a well-known local search heuristic which often produces good solutions for balanced cut, i.e. cuts which are close to minimal. This problem introduces the KL algorithm and asks you to provide an efficient implementation.

Given a weighted undirected graph $G(V, E)$ with $2n$ nodes, the KL algorithm begins by arbitrarily partitioning V into sets A and B, each with n nodes. KL then tries to improve this partition, by repeatedly finding nodes $a \in A$ and $b \in B$ such that *swapping* a and b, i.e. moving a to B and b to A, reduces the weight of the cut. It creates a new partition formed by the best subset of these swaps. This process is repeated until no subset of swaps improves the cut, i.e. we have discovered a local minimum, or some maximum runtime has been exceeded.

In more detail, for each $a \in A$, let I_a be the internal cost of a, that is, the sum of the weights of edges between a and other nodes in A, and let E_a be the external cost of a, that is, the sum of the weights of edges between a and nodes in B. Similarly, define I_b, E_b for each $b \in B$. Furthermore, let

$$D_s = E_s - I_s \tag{1}$$

be the difference between the external and internal costs of side s, for $s \in \{A, B\}$. If a and b are swapped, then the reduction in cost is

$$T_{old} - T_{new} = D_a + D_b - 2c_{a,b} \tag{2}$$

where $c_{a,b}$ is the weight of the possible edge between a and b.

The KL algorithm attempts to find an optimal series of swaps between elements of A and B which maximizes $T_{old} - T_{new}$ and then executes these operations, producing a new partition of the graph. This process is repeated until the cut cannot be improved, or a maximum runtime is exceeded. The pseudocode is shown below. In practice, the repeat loop is typically executed a constant number of times and produces a small cut.

Describe an efficient implementation of the KL algorithm. If the repeat loop is run k times, analyze the time complexity of your algorithm.

Algorithm 2. Problem 6

```
procedure KERNIGHAN-LIN(A,B)

    repeat
        $A_0 = A$; $B_0 = B$
        Compute $D$ values for all $a \in A_0$ and all $b \in B_0$

        for $i := 1$ to $n$ do
            Find an allowed $a_i \in A_{i-1}$ and $b_i \in B_{i-1}$ which maximizes $g_i = D_{a_i} + D_{b_i} - 2c_{a_i b_i}$
            Set $A_i = A_{i-1} - \{a_i\} \cup \{b_i\}$ and $B_i = B_{i-1} - \{b_i\} \cup \{a_i\}$
            Remove $a_i$ and $b_i$ from further consideration, i.e. subsequent iterations of the for loop are not allowed to choose $a_i$ or $b_i$ again
            Update $D$ values for all nodes in $A_i, B_i$

        Find $k$ which maximizes $g_{max} = \sum_{i=1}^{k} g_i$
        if $g_{max} > 0$ then
            Swap $a_1, a_2, ..., a_k$ with $b_1, b_2, ..., b_k$ to form the new partition $A, B$
    until $g_{max} \leq 0$

end
```