CS 3510 Homework 1

Wenqi He

September 6, 2017

1

(a)

By definition of big-O notation, if $f(n) \leq \mathcal{O}(g(n))$ and $g(n) \leq \mathcal{O}(h(n))$, then there exists some constants c_1 and c_2 such that for large enough n

$$f(n) \leq c_1 g(n)$$

$$g(n) \leq c_2 h(n)$$

Combining the two inequalities,

$$f(n) \leq c_1 g(n) \leq c_1 c_2 h(n)$$

In other words, there exists $c = c_1 c_2$ such that for large enough n

$$f(n) \leq c h(n),$$

therefore,

$$f(n) \leq \mathcal{O}(h(n))$$

(b)

The innermost loop is linear. The two outer loops are similar to the loops in insertion sort, but since i and j can only take $\Theta(\log(n))$ values, so the time complexity is $\Theta^2(\log(n))$. The time complexity of the entire nested loops is therefore

$$\Theta(\log^2(n)) \cdot \Theta(n) = \Theta(n \log^2(n))$$

(c)

The statement is false.
Let $g(n) = c \cdot f(n)$, then for all n,

$$g(n) \leq cf(n) \text{ and } g(n) \geq cf(n).$$

By definition of big-O notations, $f(n) \leq \mathcal{O}(g(n))$ and $f(n) \geq \Omega(g(n))$, which is an counterexample. Q.E.D.

(d)

The statement is true.
Since $\log^a n \leq \mathcal{O}(n^b)$ for any positive a and b,

$$\log^{10} n \leq \mathcal{O}(n^{0.1}),$$

$$\begin{aligned} n^2 \log^{10} n &\leq \mathcal{O}(n^2 \log^{10} n) \\ &= \mathcal{O}(n^2) \cdot \mathcal{O}(\log^{10} n) \\ &\leq \mathcal{O}(n^2) \cdot \mathcal{O}(n^{0.1}) \\ &= \mathcal{O}(n^{2.1}) \end{aligned}$$

Q.E.D.

(e)

The statement is false.
$\forall c \in (0, \infty)$, there exists a N large enough such that $2^N = c$, then $\forall n > N$,

$$2^{2n} = 2^n \cdot 2^n > 2^N \cdot 2^n = c \cdot 2^n$$

In other words, $2^{2n} = 2^n \cdot 2^n$ always exceeds $c \cdot 2^n$ as n goes to infinity no matter how large c is. Q.E.D.

2

(a)

$$a = 3, \quad b = 4, \quad d = 1$$

By Master Theorem, since $\log_4(3) < 1$,

$$T(n) \leq \mathcal{O}(n)$$

(b)

$$a = 8, \quad b = 4, \quad d = 1.5$$

By Master Theorem, since $\log_4(8) = 1.5$,

$$T(n) \leq \mathcal{O}(n^{1.5} \log n)$$

(c)

The recurrence relation is

$$T(n) = 3T\left(\frac{n}{4}\right) + \mathcal{O}(1)$$

$$a = 3, \quad b = 4, \quad d = 0$$

By Master Theorem, since $\log_4(3) > 0$,

$$T(n) \leq \mathcal{O}(n^{\log_4(3)})$$

(d)

The recurrence relation is

$$T(n) = 2^n T\left(\frac{n}{2}\right) + \mathcal{O}(n),$$

which is not in the form

$$T(n) = aT\left(\frac{n}{b}\right) + \mathcal{O}(n^d),$$

and therefore Master Theorem does not apply.

3

(a)

$$T(n) = 3T\left(\frac{3n}{5}\right) + \mathcal{O}(n)$$
$$= 3T\left(\frac{n}{5/3}\right) + \mathcal{O}(n)$$
$$a = 3, b = 5/3, d = 1$$

Since

$$3 > \frac{5}{3},$$
$$\log_{\frac{5}{3}}(3) > 1$$

By Master Theorem,

$$T(n) \leq \mathcal{O}(n^{\log_{\frac{5}{3}}(3)})$$

(b)

For example:

$$1, 4, 5, 2, 3$$

After the first sort,

$$1, 4, 5, 2, 3$$

After the second sort,

$$1, 4, 2, 3, 5$$

After the last sort,

$$1, 2, 4, 3, 5$$

The sequence is not sorted when the algorithm terminates.

4

(a)

The coefficient of the i th term is

$$\begin{aligned} c_i &= \sum_{\max\{0,i-n\}\le j\le \min\{i,n\}} a_j \cdot b_{i-j} \\ &\le \sum_{\max\{0,i-n\}\le j\le \min\{i,n\}} n^2 \\ &= \big(\min\{i,n\} - \max\{0,i-n\}\big)\cdot n^2 \end{aligned}$$

Since $\min\{i,n\} \le n$ and $\max\{0,i-n\} \ge 0$

$$\min(i,n) - \max(0,i-n) \le n$$

Therefore,

$$c_i \le n \cdot n^2 = n^3$$

By definition of big-O notation,

$$c_i \le \mathcal{O}(n^3)$$

Because all coefficients of $p(x)$ and $q(x)$ are non-negative, the sum must also be non-negative, therefore

$$c_i \in [0, \mathcal{O}(n^3)]$$

(b)

A polynomial of degree n (suppose n is even) can be split into two halves

$$\begin{aligned} p(x) &= (a_0 + \cdots + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1}) + (a_{\frac{n}{2}}x^{\frac{n}{2}} + \cdots + a_n x^n) \\ &= (a_0 + \cdots + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1}) + (a_{\frac{n}{2}} + \cdots + a_n x^{\frac{n}{2}})x^{\frac{n}{2}} \\ &= (a_0 + \cdots + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1} + 0\cdot x^{\frac{n}{2}}) + (a_{\frac{n}{2}} + \cdots + a_n x^{\frac{n}{2}})x^{\frac{n}{2}} \\ &= p_0(x) + p_1(x)x^{\frac{n}{2}} \end{aligned}$$

where $p_0(x), p_1(x)$ are both degree $\frac{n}{2}$ polynomials.
By analogy with integer multiplication,

$$\begin{aligned} p(x)\cdot q(x) &= (p_0(x) + p_1(x)x^{\frac{n}{2}})(q_0(x) + q_1(x)x^{\frac{n}{2}}) \\ &= [p_0(x)q_0(x)] + [p_0(x)q_1(x) + p_1(x)q_0)]\cdot x^{\frac{n}{2}} + [p_1(x)q_1(x)]\cdot x^n \end{aligned}$$

To compute $p(x)\cdot q(x)$, first compute:

1. $z_0 = p_0(x)q_0(x)$
2. $z_1 = p_1(x)q_1(x)$

3. $z_2 = [p_0(x) + p_1(x)] \cdot [q_0(x) + q_1(x)]$

Then

$$p(x)q(x) = z_0 + (z_2 - z_0 - z_1) \cdot x^{\frac{n}{2}} + z_1 \cdot x^n$$

Assuming integer arithmetics are $\mathcal{O}(1)$ operations, then the addition in step 3 take $\mathcal{O}(n)$ time. The last step produces a polynomial of degree $2n$, so the addition also takes $\mathcal{O}(n)$ time. The runtime recurrence for this algorithm is

$$T(n) = 3T\left(\frac{n}{2}\right) + \mathcal{O}(n)$$

Since $\log_2(3) \approx 1.58 > 1$, by Master Theorem,

$$T(n) \leq \mathcal{O}(n^{\log_2(3)}) \leq \mathcal{O}(n^{1.6})$$

Q.E.D.

(c)

The coefficient of the x^{n-s} term is

$$\begin{aligned}
&\sum_{\max\{0,-s\} \leq j \leq \min\{n,n-s\}} a_j \cdot b_{n-s-j} \\
&= \sum_{0 \leq j \leq n-s} y_j \cdot z_{n-(n-s-j)} \\
&= \sum_{j=0}^{n-s} y_j \cdot z_{s+j}
\end{aligned}$$

Q.E.D.

CS 3510 Homework 2

Wenqi He, whe47

October 9, 2017

1

(a)

1. State: $LOS[i], LES[i]$: length of the longest odd/even sequence ending at i
2. Base case: $LOS[i] \geq 1, LES[i] \geq 0$
3. Transition:

$$LES[i] = \max_{1 \leq j < i, A[j] < A[i]} \{LOS[j] + 1\}$$

$$LOS[i] = \max_{1 \leq j < i, A[j] > A[i]} \{LES[j] + 1\}$$

$$LAS[i] = \max\{LES[i], LOS[i]\}$$

(b)

Suppose there are two lists of vertices A_{even} and A_{odd}, and each element $A[i]$ is represented by vertices $A_{even}[i]$ and $A_{odd}[i]$. Every path must start from some vertex in A_{odd}, and each edge $u : i \to j$ must satisfy $i < j$ and either:
(i) the i-th vertex is in A_{odd}, the j-th vertex is in A_{even}, and

$$A_{odd}[i] < A_{even}[j]$$

(ii) the i-th vertex is in A_{even}, the j-th vertex is in A_{odd}, and

$$A_{even}[i] > A_{odd}[j]$$

This graph is a directed acyclic graph, so the longest path problem can be solved using dynamic programming in the same way as (a)

2

(a)

The cut given by $\{A, B\}$,

$$E(\{A, B\}, V \setminus \{A, B\})$$

(b)

If an edge e is in some MST in G, then removing the edge from the MST partitions the vertices in G into two sets, S and $V(G) - S$. By the cut rule, e must be the smallest edge on the cut S. If the weight of e is unchanged, and other edges either remain the same weight or gains more weight, e must still be the smallest edge on the cut S in H, and by the cut rule it must still be in some MST in H

3

(a)

Graph: A string of vertices, with each vertex of at most degree 2.
Order: From the further end to the closer end.

(b)

For such graphs there exists a shortest path tree. Suppose such a tree is already known, then we can order the edges in the following way: First update the edges that are directly connected to the root on the shortest path tree, then update the edges that are one edge away from the root, and so on. This way, when $dist[t]$ is being updated, $dist[s]$ of it's 'parent vertex' s on the shortest path tree has already been updated to its true distance, so $dist[t]$ will be updated to the true distance in the same iteration.

(c)

Using the ordering given in the problem (top to bottom, left to right), for each iteration, the true distances can propagate all the way to the right and to the bottom along some shortest path, but only 1 block leftwards or upwards, therefore the number of iterations is bounded by the length of the longest subpath in the shortest path tree that goes leftwards or upwards. To construct a graph that contains such a shortest path tree, we can:

(1) Choose path

$$\big((1,1)\cdots(1,k)\cdots(k,k)\cdots(k,k-\lfloor k/10\rfloor)\big)$$

as the shortest path to $(k, k-\lfloor k/10\rfloor))$. The subpath

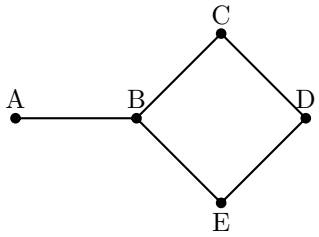
$$((1,1)\cdots(1,k)\cdots(k,k),(k,k-1))$$

takes only one iteration to update, and the rest of the path takes $\lfloor k/10\rfloor - 1$ iterations.
(2) Randomly select other branches for the shortest path tree.
(3) Set the weight of all edges on the tree to 1.
(4) Set the weight of all other edges to some sufficiently large number so that the tree constructed is indeed a shortest path tree.

Since it takes $\lfloor k/10\rfloor$ iterations to ensure that this particular shortest path is fully updated, the entire algorithm takes at least $\lfloor k/10\rfloor$ iterations.

4

(a)



Using the given algorithm, one might first color $\{A, D\}$ with color 1, then color $\{C, E\}$ with color 2, and finally color B with color 3. However, there is a way to color the graph using only 2 colors: color $\{B, D\}$ with color 1 and $\{A, C, E\}$ with color 2.

(b)

1. State: $Min(S)$: The minimum number of colors needed to color a subset S of vertices in graph G. For a graph that has n vertices, there are 2^n subsets of vertices, and therefore 2^n states.
2. Base case: $Min(\{v_i\}) = 1$
3. Transition:
Iterate through all the subsets S_i of S, if the subset is an independent set, i.e. if $Min(S_i) = 1$, compare and update $Min(S)$ by:

$$Min(S) = \min\{Min(S), 1 + Min(S - S_i)\}$$

In other words, the transition is:

$$Min(S) = \min_{1 \le i \le 2^{\|S\|}} \{1 + Min(S - S_i)\}$$

where S_i is a independent subset of S.

There are $2^{\|S\|} \le \mathcal{O}(2^n)$ subsets of S, and therefore $\mathcal{O}(2^n)$ operations are needed for each state transition. Since both the inner loop and the outer loop are $\mathcal{O}(2^n)$, the entire algorithm is $\mathcal{O}(2^n) \cdot \mathcal{O}(2^n) = \mathcal{O}(4^n)$

(c)

Since the number of operations performed on each subset S is $2^{\|S\|}$ rather than 2^n, the upper bound can actually be tighter. There are $\binom{i}{n}$ subsets of size i, and size i ranges from 0 to n, so the total runtime is:

$$T(n) = \sum_{0 \le i \le n} \binom{i}{n} \cdot 2^i = \sum_{0 \le i \le n} \binom{i}{n} \cdot 2^i \cdot 1^{n-i} = (2+1)^n = 3^n \le \mathcal{O}(3^n)$$

CS 3510 Homework 3

Wenqi He, whe47

November 5, 2017

1

(a)

It's equivalent to

$$x_2 \geq x_1$$

$$x_2 \geq -x_1$$

(b)

Let $c = |a + b - 123|$ and $d = |4a + 3b - 234|$.
The first equation is equivalent to minimizing c subject to the constraint

$$c \geq |a + b - 123|$$

Similarly, the second equation is equivalent to minimizing d subject to

$$d \geq |4a + 3b - 234|$$

Combining these contraints and the conclusion from (a), the linear program is:

Minimize: $c + d$
Subject to:

$$c \geq a + b - 123$$

$$c \geq -a - b + 123$$

$$d \geq 4a + 3b - 234$$

$$d \geq -4a - 3b + 234$$

$$5a + 4b = 345$$

2

Minimize: $a - 10c$
Subject to:

$$a + 5b - c \leq 10$$

$$-a - 5b + c \leq -10$$

$$-5b + 4d \leq 2$$

Expressing each variable as a difference of two non-negative variables, it can be written as :
Minimize $a_+ - a_- - 10c_+ + 10c_-$:
Subject to:

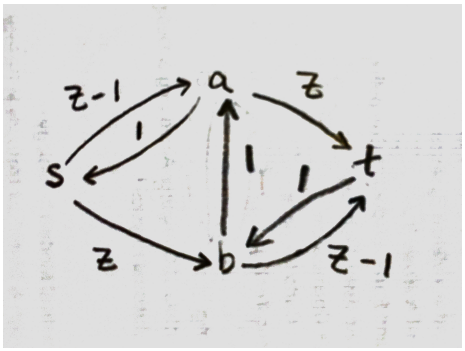
$$a_+ - a_- + 5b_+ - 5b_- - c_+ + c_- \leq 10$$

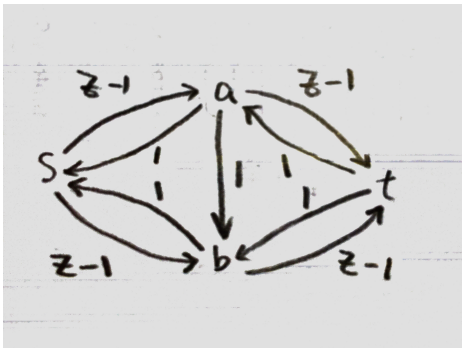
$$-a_+ + a_- - 5b_+ + 5b_- + c_+ - c_- \leq -10$$

$$-5b_+ + 5b_- + 4d_+ - 4d_- \leq 2$$

$$a_+, a_-, b_+, b_-, c_+, c_-, d_+, d_- \geq 0$$

3

(1) 

(2) 

After every 2 such iterations, the flow on path $s \to a \to t$ and $s \to b \to t$ each increases by 1, but the flow on edge $a \to b$ remains 0 because it was used twice in opposite directions. Following this algorithm, the maximum flow will be achieved after exactly $2Z$ iterations.

4

(a)

Suppose the vertices in graph G can be partitioned into A and B. For any $S_A \subset A$, suppose it's only incident to vertices in $S_B \subset B$, where $|S_B| < |S_A|$. Since the graph is d-regular, $d|S_A|$ edges come out of S_A while only $d|S_B| < d|S_A|$ edges go into S_B, therefore there must be some edges going from S_A to $B \setminus S_B$, which gives us a contradiction.

Thus, there does not exist any $S_A \subset A$ that is incident to a $S_B \subset B$ that has fewer than $|S_A|$ vertices. By Hall's Theorem, this is true if and only if such a bipartite graph has a perfect matching.

(b)

First of all, we can construct a max-flow problem from the perfect-matching finding problem if we connect a source s to each vertex in A and connect each vertex in B to a sink t. Suppose the edges only go from $\{s\}$ to A, from A to B and from B to $\{t\}$, and all edges have capacity 1. Then each perfect matching corresponds to one possible selection of edges that can be used to achieve the max flow $\frac{n}{2}$

Secondly, it can be shown that in a prefect matching if one pairing is changed all others must change as well, and therefore no two perfect matchings share any common edges.

Based on the above results, we can run Ford-Fulkerson algorithm to find one configuration that achieves max flow, and then safely remove all the edges between A and B that are used, and run the algorithm again on the remaining edges. Since exactly one edge is removed from each vertex, the remaining graph is still d-regular, and therefore all the nice properties are preserved. Repeat until all edges between A and B are removed, which would take exactly d iterations. Inside each iteration, the Ford-Fulkerson algorithm is performed on $\mathcal{O}(n)$ vertices and $\mathcal{O}(nd) \leq \mathcal{O}(n^2)$edges, and the maxflow is $\frac{n}{2} \leq \mathcal{O}(n)$. If we use Bellman-Ford to find each s-t path, the runtime of each iteration would be

$$\mathcal{O}(Fmn) \leq \mathcal{O}(n \cdot n^2 \cdot n) = \mathcal{O}(n^4)$$

In total, the runtime is

$$\mathcal{O}(dn^4) \leq \mathcal{O}(n^5)$$

CS 3510 Homework 4

Wenqi He, whe47

December 4, 2017

1

(a)

```
ALGORITHM Verifier {
  counter = 0
  FOR (i = 1...m) {
    IF (clause i evaluates to TRUE) {
      counter++
    }
  }
  IF (counter == m-1) {
    RETURN YES
  } ELSE {
    RETURN NO
  }
}
```

The algorithm runs in $\mathcal{O}(m)$ time, assuming each Boolean evaluation is $\mathcal{O}(1)$.

(b)

We can simply pick

$$clause_{m+1} = x_1$$

$$clause_{m+2} = \neg x_1$$

(c)

If F has an assignment that satisfies SAT, then only $clause_{m+1}$ or $clause_{m+2}$ in F' evaluates to FALSE under such assginment, which means that F' satisfies ALMOST-SAT. On the other hand, if F does not have such assignment, then in addition to one of $clause_{m+1}$ and $clause_{m+2}$ being FALSE, there always exists other clauses that are also FALSE, which means that F' cannot satisfy ALMOST-SAT. Lastly, the construction of F' in (b) only takes constant time, which is of course polynomial.

2

(a)

MAX-2-SAT(F, p): Given a 2-SAT formula F, output whether it's possible to find an assignment that satisfies more than p clauses in F.

(b)

Use x_i to denote whether vertex $v_i \in S$, then we can construct a 2-SAT formula as follows:

$$F = \bigwedge_{1 \le i,j \le |V(G)|} \big((x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)\big)$$

For each edge that crosses cut S, x_i and x_j have different values, therefore both $x_i \vee x_j$ and $\neg x_i \vee \neg x_j$ evaluate to TRUE.

However, if both v_i and v_j are in S, then $x_i \vee x_j = TRUE$, but $\neg x_i \vee \neg x_j = FALSE$. Similarly, if both v_i and v_j are in $G \setminus S$, then $x_i \vee x_j = FALSE$, $\neg x_i \vee \neg x_j = TRUE$.

Thus, the statement that there are k edges leaving S is equivalent to the statement that the number of clauses that are satified in F is:

$$N(\text{edges not on the cut}) + 2 \times N(\text{edges on the cut})$$

$$= (N - k) + 2k = N + k$$

Therefore, the reduction MaxCut $\rightarrow$ Max2Sat is

$$f_{\text{MaxCut} \rightarrow \text{Max2Sat}}(G, k) = (F_G, |E(G)| + k)$$

where

$$F_G = \bigwedge_{1 \le i,j \le |V(G)|} \big((x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)\big),$$

(c)

Since MAX-CUT is reducible to MAX-2-SAT, and MAX-CUT is already NP-hard (by definition of NP-complete), MAX-2-SAT must be NP-hard. It's also in NP because given the desired assignment, it only takes polynomial time to evaluate all m clauses to verify that the assginment indeed satisfies more than p clauses in F.

Because MAX-2-SAT is both NP-hard and NP, by definition, it is NP-complete.

(d)

First, we can reduce Max-2-Sat to Max-Cut using the construction provided here: `https://courses.engr.illinois.edu/cs579/sp2009/assignments/hw-1.pdf` (See Assignment 1, Problem 8). This algorithm runs in $\mathcal{O}(m)$ time and the resulting graph has $\mathcal{O}(m)$ edges. Then we can run the 2-approximation for MaxCut on this constructed graph, which will give us the 2-approximation of Max-2-Sat.

3

(a)

We can simply perform BFS on each vertex to find out all $DIST(u, v)$ and then pick the largest one. Since there are n vertices, this will take $\mathcal{O}(mn)$ time.

(b)

Suppose a and b are the most distant pair of vertices in this graph, then according to the triangular inequality,

$$\begin{aligned} DIST(a, b) &\leq DIST(a, s) + DIST(s, b) \\ &\leq \max_u DIST(s, u) + \max_u DIST(s, u) \\ &= 2 \max_u DIST(s, u) \end{aligned}$$

(c)

Randomly pick one vertex as the starting vertex s, perform BFS to compute $DIST(s, u)$ for all u, and then search for the vertex u^* with largest distance. From the result of (b),

$$DIST(s, u^*) = \max_u DIST(s, u) \geq \frac{1}{2} D,$$

which gives us the desired output s and u^*. BFS runs in $\mathcal{O}(m)$ time, and searching runs in at most $\mathcal{O}(m)$ time (in the case of linear search), so the algorithm runs in $\mathcal{O}(m)$ time.