CS170–Spring 2019 — Homework 8 Solutions

Ran Liao, SID 3034504227

March 13, 2019

Collaborators:Jingyi Xu, Renee Pu

1 Study Group

Name	SID
Ran Liao	3034504227
Jingyi Xu	3032003885
Renee Pu	3032083302

2 A Dice Game

(a)

$$W(x, y, z) = \max\{\frac{1}{6}W(x+1, y, 0)\}$$

(b) (i) **Main Idea**

Define subproblem $L[i]$ representing the minimum number of beams that can be welded into a T-foot beam. And we have the following recursive relation. And let $T[0] = 0$.

$$L[i] = \min_{j=1}^{k}\{L[i - c_j] + 1\}$$

(ii) **Proof of Correctness**

The base case is obviously correct. Then, suppose we want to compute $L[i]$ and, $L[j]$ (where $j < i$) are optimal solutions. We use the minimum value of $L[j]$ for all possible j. This is actually greedy approach and thus guarantee the optimality of $L[i]$. Therefore, $L[T]$ will be optimal solution.

(iii) **Runtime Analysis**

T iterations is needed to solve T subproblems and each of them will cost $O(k)$ time. Therefore, the overall runtime is $O(kT)$ and $O(T)$ space is needed.

3 Egg Drop

(a) Drop the first egg from 14th story. If it doesn't break, drop it from $(14+13)$th story. If still it doesn't break, drop it from $(14+13+12)$th story. Repeat this process until the first egg breaks, and then use the second egg test remaining possible range one by one in increasing story order.

14 drops is needed for this strategy. Actually, it doesn't depend on when the first egg breaks,

(b) (i) **Main Idea**

Define subproblem $F[i][j]$ representing the number of drops needed for a i story building and have j eggs. The recursive relation is:

$$F[i][j] = \min_{x=1}^{i}\{\max\{F[i-x][j]+1, F[x-1][j-1]+1\}\}$$

(ii) **Proof of Correctness**

There're two possible outcomes when dropping egg from xth story. If it doesn't break, the solution is between $(x+1)$th story and ith story. It can be considered as a new building with $i-x$ stories and can be represented like this $F[i-x][j]+1$. If it breaks, the solution is between 1st story and $(x-1)$th story. So it can be represented by $F[x-1][j-1]+1$. The max operator will make sure we consider the worst case. The min operator will give the optimal strategy.

(iii) **Runtime Analysis**

There're nk subproblems and each of them will cost $O(n)$ time. Therefore, the overall runtime is $O(n^2k)$.

4 Knightmare

(a) **Main Idea**

First sort all points into clockwise order and denote $D[i][j]$ as the distance between ith point and jth point, i.g. $D[i][j] = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

Define subproblem as $T[i][j]$, which represents the optimal solution to triangulate polygon defined by ith point to jth point. The recursive relation is as follows. In base case, if $j - i < 3$, $T[i][j]$ is 0.

$$T[i][j] = \min\{\min_{k=i+2}^{j-2}\{T[i][k]+T[k][j]+D[i][k]+D[j][k]\}, T[i][j-1]+D[i][j-1], T[i+1][j], +D[i+1][j]\}$$

And the final solution is $T[1][n]$.

(b) **Proof of Correctness**

In base case, triangle do not need to be triangulated, so the cost is 0. In induction step, for $T[i][j]$, we have $j - i - 1$ ways to choose an vertex k, so that breaks it into two or one subproblems. One subproblem is triangulate polygon defined by ith point to kth point. The other is triangulate polygon defined by kth point to jth point. Choose the smallest cost among them can guarantee the optimality of $T[i][j]$. Therefore the final answer $T[1][n]$ is optimal.

(c) **Runtime Analysis**

There're n^2 subproblems and each of them will cost $O(n)$ time to check each possible situation. Therefore, the overall runtime is $O(n^3)$.

5 Triangulating a polygon

(a) **Main Idea**

First sort all points into clockwise order and denote $D[i][j]$ as the distance between ith point and jth point, i.g. $D[i][j] = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

Define subproblem as $T[i][j]$, which represents the optimal solution to triangulate polygon defined by ith point to jth point. The recursive relation is as follows. In base case, if $j - i < 3$, $T[i][j]$ is 0.

$$T[i][j] = \min\{\min_{k=i+2}^{j-2}\{T[i][k]+T[k][j]+D[i][k]+D[j][k]\}, T[i][j-1]+D[i][j-1], T[i+1][j], +D[i+1][j]\}$$

And the final solution is $T[1][n]$.

(b) **Proof of Correctness**

In base case, triangle do not need to be triangulated, so the cost is 0. In induction step, for $T[i][j]$, we have $j - i - 1$ ways to choose an vertex k, so that breaks it into two or one subproblems. One subproblem is triangulate polygon defined by ith point to kth point. The other is triangulate polygon defined by kth point to jth point. Choose the smallest cost among them can guarantee the optimality of $T[i][j]$. Therefore the final answer $T[1][n]$ is optimal.

(c) **Runtime Analysis**

There're n^2 subproblems and each of them will cost $O(n)$ time to check each possible situation. Therefore, the overall runtime is $O(n^3)$.

6 Three Partition

(a) **Main Idea**

Define subproblem as $X[i][s_1][s_2]$, which represents whether it's possible to divide first i numbers into three groups, such that the sum of first group is s_1 and the sum of second group is s_2. Suppose the ith number is $A[i]$. The recursive relation is as follows:

$$X[i][s_1][s_2] = X[i-1][s_1][s_2] \vee X[i-1][s_1 - A[i]][s_2] \vee X[i-1][s_1][s_2 - A[i]]$$

And the final solution is $X[n][\frac{total}{3}][\frac{total}{3}]$.

(b) **Proof of Correctness**

In base case, $X[1][0][0]$, $X[1][A[1]][0]$ and $X[1][0][A[1]]$ are true. This is trivial. In induction step, for $X[i][s_1][s_2]$ we have three choices, i.g. put ith number into first group, second group or third group. $X[i-1][s_1][s_2]$ is the situation that put it into first group. $X[i-1][s_1 - A[i]][s_2]$ is the situation that put it into second group. $X[i-1][s_1][s_2 - A[i]]$ is the situation that put it into third group. So $X[i][s_1][s_2]$ will be optimal and the final answer will be optimal.

(c) **Runtime Analysis**

There're $n(\sum a_i)^2$ subproblems and each of them will cost $O(1)$ time. Therefore, the overall runtime is $O(n(\sum a_i)^2)$.

7 2-SAT

(a) if G_I has a strongly connected component containing both x and $\neg x$ for some variable x. Then there's a path from x to $\neg x$ and a path from $\neg x$ to x. The edges in this graph can be considered as implication, so we have $x \Rightarrow \neg x$ and $\neg x \Rightarrow x$ by transitive rule. So if x is assigned with true, we have contradiction $true \Rightarrow false$. If x is assigned with false, we have $true \Rightarrow false$ as well. Therefore this problem has no valid solution.

(b) Note that the clause $(\alpha \vee \beta)$ is equivalent to $(\neg\alpha \Rightarrow \beta) \wedge (\neg\beta \Rightarrow \alpha)$. The edges added in to graph G_I is symmetric.

I want to prove there's no path between SCCs that contains a literal and its negation. Suppose variable x is in SCC A and variable $\neg x$ is in SCC B. And there's a path from x to $\neg x$ that go through vertices $x, v_1, v_2, \cdots v_n, \neg x$. By symmetric property, there exists edges that from x to $\neg v_n$, from $\neg v_n$ to $\neg v_{n-1}$, from $\neg v_n$ to $\neg v_{n-1} \cdots$, $\neg v_2$ to $\neg v_1$ and from $\neg v_1$ to $\neg x$. Therefore, variable v_i and $\neg v_i$ are in the same SCC which contradicts the assumption that none of G_I 's strongly connected components contain both a literal and its negation.

Therefore, assign valid value in sink SCC will not affect the assignment of other SCCs. No contradiction will be introduced. And the method mentioned in this question is the easily way to make assignments.

(c) So first compute SCCs (meta-graph G_M of G_I) and assign value according to the previous mentioned way. Compute meta graph will cost $(O(|E| + |V|))$ time and assign will cost linear time. So the total runtime is linear.