CSC373 Homework 2

Zhiyuan Lu, Ziyi Zhou

February 28, 2022

Question 1 Investment Strategy (CLRS Exercise 15-10)

(a) Suppose for contradiction that there does not exist an optimal strategy that in each year, puts all money into a single investment.
Consider an optimal strategy S*. Without loss of generation, assume it invest d_1 dollars in investment i_i, and invest d_2 dollars in investment i_2 in year j_1, and from j_1 to j_n years the money hasn't been switched. Assume compound interests.
Then the profit for i_1 is $d_1(\prod_{j=j_1}^{j_n}(1+r_{i_1 j})-1)$, and the profit for i_2 is $d_2(\prod_{j=j_1}^{j_n}(1+r_{i_2 j})-1)$. Without loss of generation, assume $(r_{i_1 j}\prod_{j=j_1}^{j_n}(1+r_{i_1 j})-1) \geq (\prod_{j=j_1}^{j_n}(1+r_{i_2 j})-1)$. Then we can invest all d_1+d2 dollars into investment i_i to win more or equivalent profit, while reduces the number of investment from year j_1 to j_n by 1.
Therefore, by repetitively doing this way, we can reduce optimal strategy S* to only involve one investment each year. This is a contradiction. Thus, there exists an optimal strategy that in each year, puts all money into a single investment.

(b) Consider an optimal solution OPT. In year j that $j < n$, there are two options regarding how to invest in the following year $j+1$:
Option 1. Do not move the money, pay a fee of f_1 dollars.
Option 2. Shift money to other investments by either shifting money between existing investments or to a new investment, pay a fee of f_2 dollars.
OPT is the best of the two options that gives the most return from year $j+1$ to n.
Therefore the problem of planning optimal investment strategy exhibits optimal substructure.

(c) Let $OPT(j,i) =$ max return from year 1 to j, and choose investment i at year j.
Base Case: at year 1, choose i that maximizes r_{i1}, and thus the max return is r_{i1}.
Two cases regarding year j:
1.optimal return is that money is not moved in year j, which means this year's investment will be the same as previous year which is i_{j-1}.
2.optimal return is that money is moved in year j. Choose this year's investment from $i = 1,..,n, i \neq i_{j-1}$ that gives the maximum return for year j.
Bellman equation:
$OPT(j,i) = \max\{OPT(j-1,i_{j-1})(1+r_{i_{j-1}j})-f_1, OPT(j-1,i_{j-1})(1+\max_{i\neq i_{j-1}}(r_{ij}))-f_2\}$

Algorithm 1 Store maximum return at end of each year and the investment at each year to OPT

1: **procedure** StoreInvest($d, n, i_1, .., i_n, r_{11}, ...r_{n10}, f1, f2$)
2: Initialize OPT as a list of zero tuples $[(0,0),(0,0),..(0,0)]$
3: $OPT[1] \leftarrow (d(1+max(r_{i1})), max\_index(r_{i1}))$
4: **for** $j = 2, ..., 10$ **do**
5: $i_{max} \leftarrow max\_index(r_{ij})$
6: $i_{j-1} \leftarrow OPT[j-1][1]$
7: $returns = [OPT[j-1][0](1+r_{i_{j-1}j})-f_1, OPT[j-1][0](1+r_{i_{max}j})-f_2]$
8: $invests = [i_{max}, i_{j-1}]$
9: **if** $i_{max} \geq i_{j-1}$ **then**
10: $idx = max\_index(returns)$
11: **else**
12: $idx = 1$
13: $OPT[j][0] \leftarrow returns[idx], OPT[j][1] \leftarrow invests[idx]$
return OPT

Then we can find the optimal investment at each year j from $OPT[j][1]$, and the maximum return at end of 10 years from $OPT[10][0]$.
In the function StoreInvest, There are 10 iterations in the for loop in total. At each iteration, line 5 cost $O(n)$, line 6 to 10 cost $O(1)$. To find the optimal investment and maximum return cost $O(1)$. Therefore the runtime in total is $O(n)$.

(d) With the additional restriction, there is no longer guaranteed that regarding each year j, there would be exactly two cases. Because if we exceed the $15,000$ in the case which gives the maximum return, we need to reconsider a best possible investment. If the reconsidered best choice also exceed the limit, we need to repeatedly do the step again.
In addition, whether to switch money at the beginning of each year is now dependent on how much money we have at the beginning of each year. If we exceed the limit, then we must switch money.

Therefore, there is no clear bound that how many sub-problems we have in each year. Thus the optimal substructure doesn't exist.

Question 2 Party Planning

(a) Let $OPT(T)$ = maximum sum of "fun" ratings of the guests from the tree T, based on the invitation rule in the problem description.
$OPT(T)$ is the quantity that dynamic programming approach computes.

(b) Bellman:

$$OPT(T) = \begin{cases} rating(root) & \text{T's root has no children} \\ \max\{\sum_{t \in child(T)} OPT(t), rating(root) + \sum_{t \in grandchild(T)} OPT(t)\} & \text{otherwise} \end{cases}$$

Justification:
Base Case: when T only has one node, then the maximum sum of "fun" rating can only be $rating(root)$.
Two cases regarding when T has subtrees:
1. We do not invite the root of T, then we find the optimal sum of "fun" ratings from those children trees.
2. We invite the root of T, then the direct children of its root cannot be invited, from the invitation rule in the problem description. We can only invite from its children of children, and find the optimal sum of "fun" ratings from those grandchildren trees.

(c) The Bellman equation in part (b) is computed in Top-Down order.
Let r is the root the whole tree of the company structure. The $R(r)$ is initialized beforehand as $rating(r)$, since from the invitation rule, the president must attend the party, and $rating(r)$ is a negative number. Other $R(v)$ are initialized as 0.
Call $ComputeRatings(r)$, then the algorithm will calculate and store the maximum ratings of tree rooted at each node v in $R(v)$, in the Top-Down order.

Algorithm 2 Store maximum ratings of tree rooted at node v in $R(v)$

```
1: procedure ComputeRatings(v)
2:     if R(v) was not initialized then
3:         if child(v) is None then
4:             R(v) = rating(v) return R(v)
5:         sum1 ← 0
6:         for c in child(v) do
7:             if R(c) was initialized then
8:                 sum1 = sum1 + R(c)
9:             else
10:                sum1 = sum1 + ComputeRatings(c)
11:        sum2 ← rating(v)
12:        for c in child(child(v)) do
13:            if R(c) was initialized then
14:                sum2 = sum2 + R(c)
15:            else
16:                sum2 = sum2 + ComputeRatings(c)
17:        R(v) ← max(sum1, sum2)
       return R(v)
```

(d) Assume the total number of people in the company is n.
time complexity:
In the first function call $ComputeRatings(r)$, in the first for-loop from line 6 to 10, there is the first set of possible recursive calls, and the maximum number of possible recursive calls in this for-loop is $O(n)$.
Similarly, in the second for-loop from line 12 to 16, the maximum number of possible recursive calls in this for-loop is $O(n)$.
Since we omit the unnecessary repeated recursive calls by storing $R(v)$ and checking their initialization, the maximum rating at each node is calculated exactly once.
Therefore, there will be maximum n function calls in total, to calculate and store all $R(v)$.
Thus, the total runtime is $O(n)$.
space complexity:
If we assume we need 1 bit to store each element in the list $R(v)$, then the algorithm need maximum $O(n)$

bits to store all information of maximum ratings at each node, in order to calculate the maximum ratings of the whole tree.

(e) Initialize L as a global list.
Call $ProduceList(r, R)$ will produce the guest list , where R is the list of maximum ratings calculated before, and r is the root of the whole tree.

Algorithm 3 Produce the guest list L from the maximum ratings given by R

procedure PRODUCELIST(v, R)
  if v is the root of whole tree or child(v) is None **then**
    $L.append(v)$ **return**
  $sum1 \leftarrow 0$
  for c in child(v) **do**
    $sum1 = sum1 + R(c)$
  $sum2 \leftarrow rating(v)$
  for c in child(child(v)) **do**
    $sum2 = sum2 + R(c)$
  if R(v) = sum1 **then**
    for c in child(v) **do**
      $ProduceList(c, R)$
  if R(v) = sum2 **then**
    $L.append(v)$
    for c in child(child(v)) **do**
      $ProduceList(c, R)$
  return

Question 3 Doctor Scheduling

(a) Let the days be $d_1, d_2, \cdots, d_n$. Each doctor can work at most n days. Create a flow network instance with the set of vertices $\{s, t, a_1, \cdots, a_n, d_1, \cdots, d_n\}$ and the following edges:

- $\{s, a_i\}$ with capacity n, for each a_i
- $\{d_j, t\}$ with capacity a_j, for each d_j
- $\{a_i, d_j\}$ with unit capacity, for each a_i and d_j such that doctor a_i would like to work on day d_j (i.e. $d_j \in A_i$).

The algorithm is as follows:

(i) Compute a maximum flow f in the above instance via the Ford-Fulkerson algorithm, which will be an integral flow.

(ii) If $0 \leq f(s, a_i) \leq n$ for each doctor a_i, and $f(d_j, t) = a_j$ for each day d_j, then a feasible schedule exists. Assign doctor a_i to work on day d_j whenever $f(a_i, d_j) = 1$.

(iii) Otherwise, report that no schedule is possible given the doctors' availability.

(b) To prove that the algorithm is correct, we will establish a 1-1 correspondence between our integral flows f and feasible schedules, and the correspondence will be that doctor a_i work on day $d_j \Leftrightarrow f(a_i, d_j) = 1$. We will prove this in both directions.

- **Valid integral flow $\Rightarrow$ feasible schedule:** Take any feasible integral flow f and construct the corresponding schedule with the above correspondence. We want to prove that this is a feasible schedule. Each d_j must have an outgoing flow of a_j. Since the edges from doctors to days have unit capacity and the flow is integral, each d_j has a_j outgoing edges carrying a unit flow, and the rest carrying no flow. Since $a_j \leq n$ and each edge $\{a_i, d_j\}$ satisfies $d_j \in A_i$, each a_i has at most $|A_i|$ outgoing edges, and $|A_i| \leq n$. Therefore, by our correspondence, each doctor a_i only work when they are available and exactly a_j doctors work on day d_j, so the schedule is feasible.
- **Feasible schedule $\Rightarrow$ valid integral flow:** Take any feasible schedule and construct a integral flow f with the above correspondence. Since there are a_j doctors working on each day d_j, each d_j has an outgoing flow of a_j. And each doctor a_i only work on days when they are available, so each a_i has an incoming flow $l_i \leq |A_i| \leq n$. Therefore, $f(s, a_i) = n$ for each a_i and $f(d_j, t) = a_j$ for each d_j satisfies the flow conservation constraints, and the flow is valid.

Runtime:

This flow network has $n + n + 2$ vertices and $n + n + \sum_{i=1}^{n} |A_i| \leq 2n + n^2$ edges.

The sum of capacities of edges leaving s is $\sum_{i=1}^{n} l_i \leq n^2$, since $l_i \leq |A_i| \leq n$ for each i. Hence, the worst-case running time of the algorithm is $\mathcal{O}(n^4)$.

(c) Assume $c \leq n - \min_{1 \leq i \leq n} |A_i|$.

Create a flow network instance with the set of vertices $\{s, t, a_1, \cdots, a_n, a'_1, \cdots, a'_n, d_1, \cdots, d_n\}$ and the following edges:

- $\{s, a_i\}$ with capacity n, for each a_i, and $\{s, a'_i\}$ with capacity c, for each a'_i
- $\{d_j, t\}$ with capacity a_j, for each d_j
- $\{a_i, d_j\}$ with unit capacity, for each a_i and d_j such that doctor a_i would like to work on day d_j (i.e. $d_j \in A_i$).
- $\{a'_i, d_j\}$ with unit capacity, for each a_i and d_j such that $d_j \notin A_i$.

The algorithm is as follows:

(i) Compute a maximum flow f in the above instance via the Ford-Fulkerson algorithm, which will be an integral flow.

(ii) If $0 \leq f(s, a_i) \leq n$ and $0 \leq f(s, a'_i) \leq c$ for each doctor a_i, and $f(d_j, t) = a_j$ for each day d_j, then a feasible schedule exists. Assign doctor a_i to work on day d_j whenever $f(a_i, d_j) = 1$ or $f(a'_i, d_j) = 1$.

(iii) Otherwise, report that no schedule is possible given the doctors' availability.

(d) To prove that the algorithm is correct, we will establish a 1-1 correspondence between our integral flows f and feasible schedules. We will prove this in both directions.

- **Valid integral flow ⇒ feasible schedule:** Take any feasible integral flow f and construct the corresponding schedule with the above correspondence. We want to prove that this is a feasible schedule. Each d_j must have an outgoing flow of a_j. Since the edges from doctors to days have unit capacity and the flow is integral, each d_j has a_j outgoing edges carrying a unit flow, and the rest carrying no flow. Since each edge $\{a_i, d_j\}$ satisfies $d_j \in A_i$, each a_i has at most $|A_i|$ outgoing edges; and each edge $\{a'_i, d_j\}$ satisfies $d_j \notin A_i$, each a'_i has at most $(n - |A_i|)$ outgoing edges, and $n - |A_i| \leq c$.

  Therefore, by our correspondence, each doctor a_i only work when they are available or at most c days outside their availability list, and exactly a_j doctors work on day d_j, so the schedule is feasible.
- **Feasible schedule ⇒ valid integral flow:** Take any feasible schedule and construct a integral flow f with the above correspondence. Since there are a_j doctors working on each day d_j, each d_j has an outgoing flow of a_j. And each doctor a_i work on days when they are available or at most c days outside their availability list, so each a_i has an incoming flow $l_i \leq |A_i| \leq n$, and each a'_i has an incoming flow $l'_i \leq c \leq n - |A_i|$. Therefore, $f(s, a_i) = n$ for each a_i, $f(s, a'_i) = c$ for each a'_i, and $f(d_j, t) = a_j$ for each d_j satisfies the flow conservation constraints, and the flow is valid.

Runtime:

This flow network has $2n + n + 2$ vertices and $2n + n + \sum_{i=1}^{n} |A_i| + \sum_{i=1}^{n} |A - A_i| \leq 3n + n^2$ edges.

The sum of capacities of edges leaving s is $\sum_{i=1}^{n} l_i + l'_i \leq \sum_{i=1}^{n} |A_i| + n - |A_i| \leq n^2$. Hence, the worst-case running time of the algorithm is $\mathcal{O}(n^4)$.

Question 4 Exam Invigilation

(a) Let the CPOs be $CPO_1, CPO_2, \cdots, CPO_n$, and the exam slots be $e_1, e_2, \cdots, e_m$, where each exam slot e_i has l_i exams. Assume each CPO_i has a list of availability A_i, i.e. CPO_i can invigilate an exam during exam slot e_j if $e_j \in A_i$, and the maximum number of exam slots that CPO_i can invigilate is $c_i, c_i \leq |A_i|$. Since there are m exams, there are at most m days. Let $d_{i1}, d_{i2}, \cdots, d_{im}$ be the days of exams for CPO_i.

Create a flow network instance with the set of vertices $\{s, t, CPO_1, \cdots, CPO_n,$ $d_{11}, \cdots, d_{1m}, \cdots, d_{n1}, \cdots, d_{nm}, e_1, \cdots, e_m\}$ and the following edges:

- $\{s, CPO_i\}$ with capacity c_i, for each CPO_i
- $\{CPO_i, d_{ij}\}$ with capacity 2, for each CPO_i and d_{ij} such that CPO_i has availability on day j.
- $\{d_{ij}, e_k\}$ with capacity 1, for each d_{ij} and e_k such that exam e_k is on day j and CPO_i is available to invigilate e_k.
- $\{e_k, t\}$ with capacity $\lceil 1.1 l_k \rceil$, for each exam slot e_k

The algorithm is as follows:

(i) Compute a maximum flow f in the above instance via the Ford-Fulkerson algorithm, which will be an integral flow.

(ii) If $0 \leq f(s, CPO_i) \leq c_i$ for each CPO_i, and $f(e_k, t) = \lceil 1.1 l_k \rceil$ for each exam slot e_k, then a feasible schedule exists. Assign CPO_i to invigilate on exam period e_k whenever $f(d_{ij}, e_k) = 1$.

(iii) Otherwise, report that no assignment is possible given the CPOs' availability.

(b) To prove that the algorithm is correct, we will establish a 1-1 correspondence between our integral flows f and feasible assignments. We will prove this in both directions.

- **Valid integral flow ⇒ feasible assignment:** Take any feasible integral flow f and construct the corresponding assignment with the above correspondence. We want to prove that this is a feasible assignment. Each e_k must have an outgoing flow of $\lceil 1.1 l_k \rceil$. Since the edges from day per CPO d_{ij} to exam periods have unit capacity and the flow is integral, each e_k has exactly $\lceil 1.1 l_k \rceil$ incoming edges carrying a unit flow. And since the flow satisfies $0 \leq f(s, CPO_i) \leq c_i$, each CPO will not invigilate more exams than their maximum availability. Since $f(d_{ij}, e_k) \leq 2$, each CPO will invigilate at most two exams on a single day. Therefore, by our correspondence, each CPO_i only invigilate the exam periods when they are available, so the assignment is feasible.
- **Feasible assignment ⇒ valid integral flow:** Take any feasible assignment and construct a integral flow f with the above correspondence. By the construction, it is clear that f is integral. Each CPO_i only invigilate on exam slots when they are available and the number will not exceed the maximum available number of exam slots, so $f(s, CPO_i) \leq c_i$ for each CPO_i. Each CPO will invigilate at most

2 exams on each day, so $d_{ij} \leq 2$. And each exam slot has $\lceil 1.1l \rceil$ available CPOs, so it has $\lceil 1.1l \rceil$ incoming edges carrying a unit flow, so $f(e_k, t) = \lceil 1.1l_k \rceil$ for each e_k. Therefore, the flow satisfies the flow conservation constraints, and it is valid.

Runtime:

This flow network has $n + mn + m + 2$ vertices. There are n CPOs and each has at most m edges to their corresponding days. There are mn days per CPO, and each day per CPO has at most 3 edges to exam slots. So the number of edges is at most $n + mn + 3mn + m = n + m + 4mn$ edges.

The sum of capacities of edges leaving s is $\sum_{i=1}^{n} c_i \leq n^2$. Hence, the worst-case running time of the algorithm is $\mathcal{O}(mn^3)$.