Homework 3

Haoyu Zhen

April 26, 2022

Problem 1

(a). Suppose that $\{d_i\}$ is sorted. If $\exists i \in \{1, 2, \cdots, n-1\}$, $d_{i+1} - d_i > C$ or $D - d_n > C$, then it is NOT possible to reach B from A. If not, it is possible.

(b). Intuitively, I design the Algo.2 based on Algo.1. While Algo.2 may be indecipherable because I use a heap to optimize its complexity. The naive version of the backbone (the "while") is available at Algo.3.

The correctness of Algo.3 entails that of Algo.2. I.e., the use of heap just reduce many operations of line 11 in Algo.3. Now I will use mathematical induction to show the soundness of Algo.3:

- Suppose that $\boldsymbol{d}$ is ordered and $\mathcal{S}_k$ is the minimum cost from A to d_k.
- Now the car is at d_k with the corresponding gas C_{now} left in tank.
- Assume that $(d_{k+1}, d_{k+2}, \cdots, d_l) \leq C + d_k$ while $d_{l+1} > C + d_k$. For simplicity, $\mathcal{D} \triangleq \{k+1, \cdots, l\}$
- Case I: $\forall i \in \mathcal{D}$, $p_i > p_k$. The driver should fill the tank full and go to station $m \triangleq \arg\min_{i\in\mathcal{D}} p_i$. This is because in $[d_k + C_{\text{now}}, d_k + C]$, he will spend $p_k(C - C_k) < p_k l_k + \sum_{i\in\mathcal{D}} p_i l_i$ where $l_k + \sum_{i\in\mathcal{D}} l_i = C - C_{\text{now}}$.
- Case II: $\exists i \in \mathcal{D}$, $p_i \leq p_k$ while $(p_{k+1}, p_{k+2}, \cdots, p_{i-1}) > p_k$. If $C_{\text{now}} > d_i - d_k$, go to i directly beacuse $p_k \sum_t l_t < p_i \sum_t l_t$ (station i could cover any point k could reach). If not, fill the tank just to i (i.e., fill the gas tank to $d_i - d_k$). This holds by

  $$p_k(C_{\text{now}} + d_i - d_k) + p_k \sum_t l_t < \sum_{t=k}^{i-1} p_t l_t + p_i \sum_t l_t$$

  where $\sum_{t=k}^{i-1} l_t = C_{\text{now}} + d_i - d_k$.

The complexity of Algo.2: Every stataion is inserted and poped **once** in the heap. So

$$T(n) = \mathcal{O}(n \log n).$$

For *naive* version, the complexity is $\mathcal{O}(n^2)$.

Algorithm 1 $\mathcal{GO}(l, p_k, d_k, d_l, \mathcal{S}, C_{\text{now}})$

Input: The car is at station k with gas C_{now} and cost $\mathcal{S}$. He want to go to station l directly.
Output: Update the value of $\mathcal{S}, C_{\text{now}}$ at station l.

```
1: S ← S + p_k max(d_l − d_k − C_now, 0)
2: C_now ← C_now + max(d_l − d_k − C_now, 0) − (d_l − d_k)
3: return (l, S, C_now)
```

Algorithm 2 Minimize the gas cost

Input: Distance D, capacity C, and 2 sequence $\boldsymbol{d} = \{d_i\}$, $\boldsymbol{p} = \{p_i\}$ where $i = 1, 2, \cdots, n$

Output: Minimal cost: $\mathcal{S}$

```
d_{n+1} = D,  p_{n+1} = +∞ and d ← d ∪ {d_{n+1}},  p ← p ∪ {p_{n+1}}
Sort (d_i, p_i) w.r.t. d_i such that d_i < d_{i+1} for all i.
k ← 1, l ← 2 and C_now ← 0
Initialize a min-heap H to store (i, d_i, p_i).
In H, (i, d_i, p_i) < (j, d_j, p_j) if p_i < p_j. Also, top(H) will return the index instead of d_i or p_i.
Insert (1, d_1, p_1) into H
while k < n + 1 do
    flag←False
    while d_l − d_k < C do
        if p_l ≤ p_k then
            k, S, C_now ← GO(l, p_k, d_k, d_l, S, C_now)          #Go to l directly
            flag←True
            Break
        end if
        Insert (l, d_l, p_l) into H
        l ← l + 1
    end while
    if flag then
        Continue                                                   #We have changed k
    end if
    while top(H) < k and size(H) > 0 do
        H.pop()                                                    #Pop the stations in front of k
    end while
    if size(H) < 0 then
        return IMPOSSIBLE                                          #We could not reach any station at k
    end if
    j ← top(H) and H.pop()
    if p_j ≤ p_k then
        k, S, C_now ← GO(j, p_k, d_k, d_j, S, C_now)              #Go to j directly
    else
        if C + d_k > D then
            k, S, C_now ← GO(n + 1, p_k, d_k, d_{n+1}, S, C_now)  #Go to B (the destination)
        else
            S ← S + (C − C_now)p_k                                 #Fill the tank full
            C_now ← C − C_now
            k ← j
        end if
    end if
end while
```

Algorithm 3 The backbone of Algo.2

while $k < n+1$ **do**
  flag←False
  $l \leftarrow k$
  for $d_l - d_k < C$ **do**
    if $p_l < p_k$ **then**
      $k, \mathcal{S}, C_{\text{now}} \leftarrow \mathcal{GO}(l, p_k, d_k, d_l, \mathcal{S}, C_{\text{now}})$ *#Go to l directly*
      flag←True
      Break
    end if
    if $p_m \geq p_l$ **then**
      $m = l$ *#Find the minimum of p_l station k could reach*
    end if
    $l \leftarrow l + 1$
  end for
  if flag **then**
    Continue
  end if
  $k, \mathcal{S}, C_{\text{now}} \leftarrow \mathcal{GO}(m, p_k, d_k, d_m, \mathcal{S}, C_{\text{now}})$ *#Go to m directly*
end while

Problem 2

(1).

Algorithm 4 $\mathcal{M}(T)$

Input: A given tree $T = (V, E)$ with the root r
Output: A minimum-size subset $\mathcal{S}$ of vertices that covers all the vertices in G w.r.t $k = 1$.

if $|V| = 1$ **then**
  return V
end if
$\mathcal{S} \leftarrow \varnothing$
Implement BFS from root r during which update the height and parent for each vertex
Meanwhile, store the vertex we meet into a stack s *#In s, vertex's height is ordered*
while s is not empty **do**
  $v \leftarrow s$.top *#v has the largest height in s*
  s.pop
  if v is not marked **then**
    if v is the root **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{v\}$
    end if
  else
    Mark v and get v's parent u
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{u\}$
    Mark u's neighbor and u itself
  end if
end while
return $\mathcal{S}$

The correctness of Algo.4:

- First I need to interpret what Algo.4 do: for given T, we find its leaves. Put these leaves' parents into $\mathcal{S}$ and delete them. And do it repeatly. I use a stack and BFS to ameliorate the complexity.
- In T, if u is a leaf whose parent is v, v must in the vertex cover (because v dominate u).
- Then deleting u and u's neighbor, we get a subgraph. It's obvious that $\mathcal{S} = \mathcal{S}' \cup \{u\}$ where $\mathcal{S}'$ is minimum and cover the subgraph

The complexity: $\mathcal{O}(|V|)$ because:

- $\mathcal{O}(\text{BFS}) = |V|$. Every vertex is pushed and poped once.
- Every vertex is marked at most twice (by "itself/one of its child" and by his parent)

(2). Similarly,

Algorithm 5 $\mathcal{M}(T)$

Input: A given tree $T = (V, E)$ with the root r
Output: A minimum-size subset $\mathcal{S}$ of vertices that covers all the vertices in G w.r.t k.

```
if |V| = 1 then
    return V
end if
S ← ∅
Implement BFS from root r during which update the height and parent for each vertex
Meanwhile, store the vertex we meet into a stack s          #In s, vertex's height is ordered
while s is not empty do
    v ← s.top                                               #v has the largest height in s
    s.pop
    if v is not marked then
        if v's height< k then
            S ← S ∪ {r}
            return S
        else
            Mark v and get v's k^th ancestor u
            S ← S ∪ {u}
            Mark every vertex t ∈ {t | d(u,t) ≤ k} where d(a,b) is the distance between a and b
        end if
    end if
end while
return S
```

The soundness of Algo.5 is similar as Algo.4.

- To generalize, I just simply alter the way of marking vertices.
- Also, for every $v' \in T$ could cover the deepest leaf v, v' is in u's subtree which contain v. And it is obvious that u dominate the vertices v' could reach.

The complexity: $\mathcal{O}(f(k)|V|)$.

- The function $f(k)$'s physical interpretation is $\max_T$(average times for which a vertex is marked in T)
- The function $f(\cdot)$ may not be explicit but $f(k) < |V|$ and $f(1) = 2$.

Now I will optimize Algo.5. Indeed, line 18 is redundant. This time I will mark vertex with a specific number m_v.

Algorithm 6 The optimized version of the "while" backbone in $\mathcal{M}(T)$

while s is not empty **do**
    $v \leftarrow s$.top *#v has the largest height in s*
    s.pop
    $u \leftarrow v$, $flag \leftarrow$True
    while $d(v, u) \le k$ **do**
        if u is marked and $d(u, v) + m_u \le k$ **then**
            $flag \leftarrow$False, **break** *#This means that v has been covered*
        end if
        $u \leftarrow u$'s parent (check before: if u is the root, **break**)
    end while
    if $flag$ **then**
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{u\}$
        Mark every vertex $t \in \{t \mid d(u, t) \le k \wedge t$ is the ancestor of $v\}$ and $m_t = \min[d(u, t), m_t]$
        #We initiate $m_t = +\infty$ for every $t \in V$
    end if
end while
return $\mathcal{S}$

Correctness:

- I use a trick to determine whether a vertex has already been covered.
- That is: if his ancestor u is marked with a number m. This number tells us that $d(u, t)$ where t is in $\mathcal{S}$.
- If $d(u, t) + d(u, v) \le k$, vertex v could be covered by t.

The complexity is $\mathcal{O}(k|V|)$ because every vertex will be marked and checked for at most $2k + 1$ times.

Problem 3

(a). Let set A and B be 2 different maximal independent sets of $\mathcal{I}$. If $|A| < |B|$, then $A \subset A \cup \{x\}$ where $x \in B - A$. So it holds that $|A| = |B|$

(b).

Definition 1. *A is a set containing edges. $\mathcal{F}_A \triangleq (V_A, A)$ where $V_A \triangleq \{v|(v, u) \in A$ for some $u\}$.*

Lemma 1. *For all $A \in \mathcal{I}$, $\mathcal{F}_A$ is a forest.*

By lemma and defination above,

- Trivially, A is hereditary. A forest's subgraph is a forest.
- Let set $A, B \in \mathcal{I}$ such that $|A| < |B|$. We have $|V_A| < |V_B|$. There exists $v \in V_B$, $v \notin V_A$. Also, $\exists e = (v, v') \in B - A$. Finnaly, $A \cup \{e\} \in \mathcal{I}$ because v in $\mathcal{F}_{A\cup\{e\}}$ is a leaf. The exchange property holds for $\mathcal{I}$.

Apparently, the maximal sets of this matroid is the maximum spanning forest of G.

(c). Firstly, $\{x\} \in \mathcal{I}$ and $\forall w(y) > w(x) : \{y\} \notin \mathcal{I}$. Suppose that $\mathcal{S}$ is the maximal independent set with maximum weight. By finite steps we could generate a maximal independent set $\mathcal{S}'$:

1. $\mathcal{S}' = \{x\}$
2. Update $\mathcal{S}'$ following exchange property: $|\mathcal{S}'| < |\mathcal{S}|$. Then $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{x'\}$ where $x' \in \mathcal{S} - \mathcal{S}'$.
3. Do step 2 repeatly and finnaly we have $|\mathcal{S}| = |\mathcal{S}'|$.

Obviously, $|\mathcal{S}' - \mathcal{S}| \leq 1$. Thus $w(\mathcal{S}') \geq w(\mathcal{S})$ because $w(x) = \arg\max_{y \in \{y | \{y\} \in \mathcal{I}\}} w(y)$. Ultimately, $\mathcal{S}'$ is what we want.

(d). Proof by mathematical induction. The abbreviation "MISMW" means the maximal independent set with maximum weight.

- The first element the algorithm added to $\mathcal{S}$ is x_1. By (c), $\exists \mathcal{S}' \in \mathcal{I}$ such that $\{x_1\} \subset \mathcal{S}'$.
- Suppose that the algorithm has added $\{x_1, x_2, \cdots, x_k\}$ into $\mathcal{S}$. And $\exists \mathcal{S}_k \in \mathcal{I}$ such that $\mathcal{S} \subset \mathcal{S}_k$.
- Now the algorithm will add x_{k+1} into $\mathcal{S}$. Some property we could conclude: $\forall y$ such that $w(x_{k+1}) < w(y)$ and $y \neq (x_1, \cdots, x_k)$, there's no MISMW containing $\{x_1, x_2, \cdots, x_k, y\}$. Then we could do step.2 in (c) repeatly with $\mathcal{S}_k$. And finally we will get $\mathcal{S}_{k+1}$ which is a MISMW and contains $\{x_1, x_2, \cdots, x_{k+1}\}$. (Nota bene: this is true by $|\mathcal{S}_{k+1} - \mathcal{S}_k| \leq 1$ and the property mentioned above.)

(e). Let $\mathcal{I} = \{u \subset U | \text{vectors in } u \text{ is linearly independent}\}$. Obviously $M = (U, \mathcal{I})$ is a matroid: M is naturally hereditary. Also if $|A| < |B|$, then $\exists \vec{v} \in B$ such tht $A \cup \{\vec{v}\}$ is linearly independent by the fact that $\dim(\text{span}(A)) < \dim(\text{span}(B))$.
So we just run the algorithm given in problem set with a tuning: check whether $S \cup \{x\}$ is linearly independent. Here's the implementation of the check:
The complexity is $\mathcal{O}(n^2)$.

Algorithm 7 $\mathcal{F}(\boldsymbol{x}, C)$: minimize the number of machines approximately

Input: A linearly independent set $S = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \cdots, \boldsymbol{s}_k\}$ and a vector $\boldsymbol{x}$
Output: True or False: $S \cup \{x\}$ is linearly independent

$m \leftarrow 1$
while $\boldsymbol{x} \neq \boldsymbol{0}$ and $m \leq k$ **do**
  $\boldsymbol{x} \leftarrow \boldsymbol{x} - (\boldsymbol{s}_m \cdot \boldsymbol{x})\boldsymbol{s}_m$ *#Part of Gram-Schmidt Process*
  $m \leftarrow m + 1$
end while
return $\boldsymbol{x} \neq \boldsymbol{0}$

Problem 4

(a). Analysis of Algo.8:

Algorithm 8 $\mathcal{F}(\boldsymbol{x}, C)$: minimize the number of machines approximately

---

Input: n jobs with sizes $x_i < C$, the capacity of machines C
Output: approximately minimum number of machine: m

$m \leftarrow 1$, $l \leftarrow 1$ and $\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_n = 0$
Sort x_i such that $x_i > x_j$ for all $i < j$
while $l \leq n$ **do**
  $j \leftarrow \arg\min_{0<i<m} \mathcal{M}_i$ *#Using a heap to put $\mathcal{M}$*
  if $\mathcal{M}_j + x_l \leq C$ **then**
    $\mathcal{M}_j \leftarrow \mathcal{M}_j + x_l$
  else
    $\mathcal{M}_m \leftarrow x_l$
    $m \leftarrow m + 1$
  end if
  $l \leftarrow l + 1$
end while
return $\sum_{i=1}^{n} 1[\mathcal{M}_i \neq 0]$

Now we want to bound ALG$\triangleq m$. For simplicity, let $\min \triangleq \min_i x_i$. If $\min > C/2$, then obviously ALG=OPT. If not:

$$(\text{ALG} - 1)(C - \min) + \min < \sum_i x_i \leq C \cdot \text{OPT},$$

which means tht

$$\text{ALG} < \frac{C \cdot \text{OPT} - \min}{C - \min} + 1 \implies \text{ALG} \leq 2\text{OPT} - 1. \tag{1}$$

The complexity is $\mathcal{O}(n \log n)$.

Reference: Approximation Algorithms, Johns Hopkins University.

(b). For simplicity and without loss of generality, let $C = 1$.
My intuition:

- If $\min \leq 1/3$. From Eq.1 we conclude ALG $< 3/2$ OPT $+ \mathcal{O}(1)$.
- If $\forall i,\ x_i > 1/3$, then we have a smaller problem. Also, there are several machines which only complete one job with size $x_i > 2/3$. Let $\mathcal{S}_1 = \{x_i | \ 2/3 \leq x_i\}$.
- Let $\mathcal{S}_2 = \{x_i | \ 1/2 < x_i < 2/3\}$ and $\mathcal{S}_3 = \{x_i | \ 1/3 < x_i < 1/2\}$. If $|\mathcal{S}_2| < |\mathcal{S}_3|$, then the problem is trivial. Because the number of machines $m = |\mathcal{S}_1| + |\mathcal{S}_3|$.
- If $|\mathcal{S}_2| > |\mathcal{S}_3|$, $m = |\mathcal{S}_1| + |\mathcal{S}_3| +$ minimum number to complete $\mathcal{S}_2$. (Then we have a smaller problem and do this repeatly)

While the performance of procedure above depends on the input's distribution. I.e., if $\boldsymbol{x}$ centers around $1/2$, we get ALG < 2OPT $- 1$ again.
So I refer to First-fit bin packing, Wikipedia to desing Algo.9 whose performance is: ALG $\leq 5/3$ OPT$+$ $\mathcal{O}(1)$. (While I could not prove this proposition.)

Algorithm 9 minimize the number of machines approximately

Input: n jobs with sizes $x_i < C$, the capacity of machines C
Output: approximately minimum number of machine: m

$S_1, S_2, S_3, S_4 \leftarrow \varnothing$, $i \leftarrow 1$
while $i \leq n$ **do**
  if $C/2 < x_i \leq C$ **then**
    $S_1 \leftarrow S_1 \cup \{x_i\}$
  end if
  if $2C/5 < x_i \leq C/2$ **then**
    $S_2 \leftarrow S_2 \cup \{x_i\}$
  end if
  if $C/3 < x_i \leq \frac{2C}{5}$ **then**
    $S_3 \leftarrow S_3 \cup \{x_i\}$
  end if
  if $x_i \leq C/3$ **then**
    $S_4 \leftarrow S_4 \cup \{x_i\}$
  end if
end while
return $\mathcal{F}(S_1, C) + \mathcal{F}(S_2, C) + \mathcal{F}(S_3, C) + \mathcal{F}(S_4, C)$

Problem 5

About 3 days. Difficulty 4. Talked with Yilin Sun and Wei Jiang.