SCSE Homework 3

1 合并果子问题

问题分析

记一堆果子 a 在进行一次合并前的权值为 w_a，则将其和另一堆果子合并后，可视为 a 在合并得到的新果子堆中对权值的贡献为 $\sqrt{2w_a}$。

可以通过简单的归纳证明得到如下引理：

引理 1. *经过一系列合并得到一堆权重为 w 的果子后，若这一系列合并中有 t_i 次合并的其中一堆包含了初始权重为 w_i 的第 i 堆果子，则第 i 堆果子对 w 的贡献为 $2^{t_i 2^{-t_i}} w_i^{2^{-t_i}}$。*

结合合并操作的性质，又可以得到以下结论：

引理 2. *在引理1中，对于将 m 堆果子$\{a_1, a_2, \cdots a_m\}$进行 $m-1$ 次合并得到的一堆果子，有：*

$$\sum_{i=1}^{m} 2^{-t_{a_i}} = 1$$

这体现了量纲的一致性。

根据以上的引理，可以得出，将 n 堆果子合并为一堆，最后得到的一堆果子的权重为：

$$\begin{aligned} W &= \prod_{i=1}^{n} 2^{t_i 2^{-t_i}} w_i^{2^{-t_i}} \\ &= (2^{\sum_{i=1}^{n} t_i 2^{-t_i}})(\prod_{i=1}^{n} w_i^{2^{-t_i}}) \\ &= (2^{\sum_{i=1}^{n} t_i 2^{-t_i}})(\prod_{i=1}^{n} w_i^{2^{-t_i}}) \end{aligned} \tag{1}$$

由此可见，给定一组 $\{t_1, t_2, \cdots t_n\}$，最终结果 W 只与初始每堆果子 w_i 对应被合并的次数 t_i 有关，因此可以用 $(t_1, t_2, \cdots t_n)$ 来唯一地描述一种合并方案。记 $\{s_1, s_2, \cdots s_n\}$ 是满足 $w_{s_1} \le w_{s_2} \le \cdots \le w_{s_n}$ 的标号；要让 W 取得最小值，（将 W 取对数后）根据排序不等式，合并方案必须满足关系：

$$t_{s_1} \le t_{s_2} \le \cdots \le t_{s_n} \tag{2}$$

现给出一种满足关系2的合并方案 T_{min}：第一次将 s_n，s_{n-1} 合并为一堆；第 $k(2 \le k \le n-1)$ 次将 s_{n-k} 与上次合并得到的堆合并。即：$n-1 = t_{s_n} = t_{s_{n-1}} = t_{s_{n-k}} + k - 1 (2 \le k \le n-1)$。

以下证明方案 T_{min} 得到的最后一堆果子的权重 W 为最小值:

证明. 任取一种满足关系2的合并方案 $T' = (t'_1, t'_2, \cdots t'_n)$，且 T' 和上述给出的 T_{min} 不同，即不满足$n-1 = t'_{s_n} = t'_{s_{n-1}} = t'_{s_{n-k}} + k - 1 (2 \leq k \leq n-1)$。

则必然存在一系列 e_i（至少一个）$\{e_1, e_2, \cdots\}$ 满足 $2 \leq e_i \leq n-2$, $|e_i - e_j| \geq 2$，使得 $t'_{s_{e_i}} = t'_{s_{e_i - 1}}$。记 e_i 中最大的为 e，则此时有：

$$\begin{cases} t'_{s_e} = t'_{s_{e-1}} \\ t'_{s_e} = t'_{s_{e+1}} = t'_{s_{e+2}} - 1 = t'_{s_{e+3}} - 2 = \cdots = t'_{s_{n-1}} - (n-e-2) \\ t'_{s_{n-1}} = t'_{s_n} \end{cases} \tag{3}$$

即对于 $s_{e-1}, s_e, s_{e+1}, \cdots s_n$ 这一部分，先将 s_n 与 s_{n-1} 合并后依次与$s_{n-2}, s_{n-3}, \cdots s_{e+1}$ 合并，再将 s_e 与 s_{e-1} 合并，再将以上得到的两个堆合并为一个堆，之后执行其余的合并。

则可以根据 T'，给出一种调整后的合并方案 $T = (t_1, t_2, \cdots t_n)$ ：同样对于 $s_{e-1}, s_e, s_{e+1}, \cdots s_n$ 这一部分，先将 s_n 与 s_{n-1} 合并后依次与$s_{n-2}, s_{n-3}, \cdots s_{e-1}$ 合并，之后执行其余的合并。则此时有：

$$\begin{cases} t_{s_e} = t'_{s_e} \\ t_{s_{e-1}} = t'_{s_{e-1}} - 1 \\ t_{s_{e+1}} = t'_{s_{e+1}} + 1,\ t_{s_{e+2}} = t'_{s_{e+2}} + 1,\ \cdots t_{s_n} = t'_{s_n} + 1 \\ t_{s_1} = t'_{s_1},\ t_{s_2} = t'_{s_2},\ \cdots t_{s_{e-2}} = t'_{s_{e-2}} \end{cases} \tag{4}$$

这两种方案最终得到的一堆果子的权值之比为：

$$\begin{aligned} \frac{W_{T'}}{W_T} &= \frac{2^{\sum_{i=1}^n t'_i 2^{-t'_i}}}{2^{\sum_{i=1}^n t_i 2^{-t_i}}} \frac{\prod_{i=1}^n w_i^{2^{-t'_i}}}{\prod_{i=1}^n w_i^{2^{-t_i}}} \\ &= \left(\frac{2^{(t_{s_{e-1}}+1)2^{-(t_{s_{e-1}}+1)}}}{2^{t_{s_{e-1}}2^{-t_{s_{e-1}}}}} \prod_{i=e+1}^n \frac{2^{(t_{s_i}-1)2^{-(t_{s_i}-1)}}}{2^{t_{s_i}2^{-t_{s_i}}}}\right) \cdot \left(\frac{w_{s_{e-1}}^{2^{-(t_{s_{e-1}}+1)}}}{w_{s_{e-1}}^{2^{-t_{s_{e-1}}}}} \prod_{i=e+1}^n \frac{w_{s_i}^{2^{-(t_{s_i}-1)}}}{w_{s_i}^{2^{-t_{s_i}}}}\right) \\ &= \left(2^{(-t_{s_{e-1}}+2)2^{-t_{s_{e-1}}+1} + \sum_{i=e+1}^n (t_{s_i}-2)2^{-t_{s_i}}}\right) \cdot \left(w_{s_{e-1}}^{-2^{-(t_{s_{e-1}}+1)}} \prod_{i=e+1}^n w_{s_i}^{2^{-t_{s_i}}}\right) \\ &> \left(2^{(-t_{s_{e-1}}+2)2^{-t_{s_{e-1}}+1} + (t_{s_{e+1}}-2)\sum_{i=e+1}^n 2^{-t_{s_i}}}\right) \cdot \left(w_{s_{e-1}}^{-2^{-(t_{s_{e-1}}+1)} + \sum_{i=e+1}^n 2^{-t_{s_i}}}\right) \\ \text{记} x = t_{s_e} : &= \left(2^{(-x+1)2^{-x} + (x-1)(\sum_{i=e+1}^{n-1} 2^{-(x+i-e)} + 2^{-(x+n-1-e)})}\right) \cdot \left(w_{s_{e-1}}^{-2^{-x} + \sum_{i=e+1}^{n-1} 2^{-(x+i-e)} + 2^{-(x+n-1-e)}}\right) \\ &= \left(2^{(-x+1)2^{-x} + (x-1)2^{-x}}\right) \cdot \left(w_{s_{e-1}}^{-2^{-x}+2^{-x}}\right) \\ &= 1 \end{aligned} \tag{5}$$

因此方案 T 较 T' 更优。

显然可见，对于 T' 的 $\{e_1, e_2, \cdots\}$，从大到小地依次对每个 e_i 执行上面的调整，最后得到的权值将单调递减，方案将不断优化，且最终得到的方案将满足 $n-1 = t_{s_n} = t_{s_{n-1}} = t_{s_{n-k}} + k - 1 (2 \leq k \leq n-1)$，该条件即为 T_{min}。

$\therefore$ 方案 T_{min} 得到的最后一堆果子的权重 W 为最小值，命题得证。 □

算法描述

如算法1。

Algorithm 1: Minimum Mergence

Input: $n, w[1..n]$

Output: $weight$

sort($w[1..n]$) $s.t.\ w[i] \leq w[i+1]$;

$merged \leftarrow$merge(n, $n-1$);

$weight \leftarrow 2\sqrt{w[n] \cdot w[n-1]}$;

for $i \leftarrow n-2$ **to** 1 **do**

  $merged \leftarrow$merge($merged$, i);

  $weight \leftarrow 2\sqrt{weight \cdot w[i]}$;

end

时间复杂度分析

首先执行一次排序。使用优秀的排序算法（如桶排序、std::sort等），时间复杂度下上界分别为$\Omega(n)$ 和 $O(n \log n)$。之后有循环执行$(n-2)$次，其余语句的执行均为常数时间复杂度，故时间复杂度为$\Theta(n)$，且远小于排序过程的时间开销。

综上，算法的时间复杂度取决于排序的时间复杂度，下上界分别为$\Omega(n)$ 和 $O(n \log n)$。

2 分蛋糕问题

问题分析

显然，若一种选取方案的不公平度最小，应满足：$\neg\exists i \in [1..n]$, 未选出$i \wedge \min\{a_{s_1}, \cdots a_{s_k}\} < a_i < \max\{a_{s_1}, \cdots a_{s_k}\}$。（若不满足，则不选已选的体积最大的蛋糕，而改选 i，改选后方案的不公平度更小。）

即：若记所有蛋糕体积从小到大依次为 $v_1, v_2, \cdots v_n$，则只可能选出体积为$\{v_m, v_{m+1}, \cdots v_{m+k-1}\}$的蛋糕。这样的方案只有$(n-k+1)$种，且每种方案的不公平度为$(v_{m+k-1} - v_m)$，因此只需选取不公平度最小的即可。

算法描述

如算法2。

时间复杂度分析

对数组排序的时间复杂度下上界分别为$\Omega(n)$ 和 $O(n \log n)$；为数组初始化的时间复杂度为$\Theta(n)$；两次循环的时间复杂度分别为$\Theta(n-k)$和$\Theta(k)$；其余语句的执行均为常数时间复杂度。故总的时间

Algorithm 2: Cake Selection

Input: $n, k, a[1..n]$

Output: $selected[1..k]$, $minUnfairness$

Var $selected[1..k]$ be an array of cake volume;
Var $index[1..n]$ be an array of cake index;
$index[1..n] \leftarrow \{1, 2, 3, \cdots n\}$;
sort($index[1..n]$) $s.t.$ $a[index[i]] < a[index[i+1]]$;
Var $minUnfairness \leftarrow +\infty$;
Var $minIndexSelected$;
for $i \leftarrow 1$ **to** $n-k+1$ **do**
    if $a[index[i+k-1]] - a[index[i]] < minUnfairness$ **then**
        $minUnfairness \leftarrow a[index[i+k-1]] - a[index[i]]$;
        $minIndexSelected \leftarrow i$;
    end
end
for $i \leftarrow 1$ **to** k **do**
    $selected[i] \leftarrow index[minIndexSelected + i - 1]$;
end

复杂度取决于排序的时间复杂度，下上界分别为$\Omega(n)$ 和 $O(n \log n)$。

3 最小生成树问题

问题分析

由于 U 中的结点在最后的生成树中为叶子结点，若将所有 U 中的结点删去，则生成树的连通性保持不变，且仍为最小生成树。

据此，可以先将 U 中的结点从 G 中删去，生成最小生成树 T'；再对于 U 中的每个结点 u，选择边权最小的边(u, v) where $v \in V - U$ 加入 T'。最后得到的生成树即为所求的最小生成树 T。

可行性判断：存在一棵这样的最小生成树 T，当且仅当 G 删去 U 中的所有结点后依然为连通图，且对于 U 中的任一结点 u，至少存在一条边(u, v) where $v \in V - U$。

算法描述

如算法3。

时间复杂度分析

判断可行性：BFS的复杂度为$O(V + E)$；循环判定 U 中结点时，可利用读入边时进行预处理的结果，每次判定为常数复杂度，共判断 $|U|$ 个点，复杂度为$O(|U|)$。因此判断可行性的时间复杂度

Algorithm 3: MST

Input: $G = (V, E), U, w[u, v]$

Output: $exist, treeEdge[1..|V| - 1], minWeight$

Var $exist$: boolean, $connected$ ←BFS($G - U$);
if $\neg connected$ **then**
    $exist$ ←false;
    EXIT;
end
forall u in U **do**
    if $\neg\exists(u, v) \in E$ $where$ $v \in V - U$ **then** `// pretreat`
        $exist$ ←false;
        EXIT;
    end
end
`// use prim algorithm to build a MST for` $G - U$;
Var $color[1..|V - U|]$ be an array with all its elements initialized with $false$;
Var $key[1..|V - U|]$ be an array with all its elements initialized with $+\infty$;
Var $pred[1..|V - U|]$ be an array with all its elements initialized with $null$;
Var $treeEdge[1..|V| - 1]$ be an array of edges;
Var v_{start} ← any vertex in $V - U$;
$key[v_{start}] \leftarrow 0$;
Var $queue$ ← new $PriorityQueue(V - U)$;
Var $minWeight \leftarrow 0$;
while $\neg queue.empty()$ **do**
    $u \leftarrow queue$.extractMin();
    if $pred[u] \neq null$ **then**
        $treeEdge$.appendEdge($u, pred[u]$);
        $minWeight \leftarrow minWeight + w[u, pred[u]]$;
    end
    $color[u] \leftarrow true$;
    forall $e = (u, v)$ $where$ $e \in E \ \wedge \ v$ in $V - U$ **do**
        if $color[v] == false \ \wedge \ w[u, v] < key[v]$ **then**
            $key[v] \leftarrow w[u, v]$;
            $queue$.decreaseKey($v, key[v]$);
            $pred[v] \leftarrow u$;
        end
    end
end
forall u in U **do**
    find v_{min} where $w[u, v_{min}] \leq \forall w[u, v]$ where $v \in V - U \ \wedge (u, v) \in E$`// pretreat`;
    $treeEdge$.appendEdge(u, v_{min});
    $minWeight \leftarrow minWeight + w[u, v_{min}]$;
end

为$O(V+E)$。

寻找生成树：先判断可行性，时间复杂度为$O(V+E)$；若可行，先对$(G-U)$进行一次堆优化的prim算法，其中执行了至多 $|V|$ 次extractMin和至多 $|E|$ 次decreaseKey，复杂度为$O((V+E)\log V)$；再循环加入 U 中的结点，可利用读入边时进行预处理的结果，每加一个点为常数复杂度，共加入 $|U|$ 个点，复杂度为$O(|U|)$。因此总时间复杂度为$O((V+E)\log V)=O(E\log V)$。

4 老城区改造问题

问题分析

小区及已有的传输电路构成了图，且能通过传输电路到达的小区构成了连通块。显然，只要政府在一个小区和发电站之间建设新的传输电路，则该小区所在连通块内的所有小区都能获得电能。

因此，对于每个连通块，需要且只需要在其中一个小区和发电站之间建设新的传输电路即可。

算法描述

如算法4。不定长数组$toBuild$中存储的是需要与发电站间建设新传输电路的小区。

时间复杂度分析

相当于对全图进行了一次BFS，因此时间复杂度为$O(n+m)$。

5 货物运输问题

问题分析

记进入每个结点 v_i 且支付完该点的代价后，应拥有的最少货物量为 g_i（从而使得能够到终点时有至少m个货物）。由于每个结点的 g_i 可由下一到达结点 j 的 g_j 推出，因此可以将问题转化为：以点 T 为源的单源最短路问题。其中：$g_T=m$ 为已知量，g_S为待求的答案（由于在起点不需要支付代价）。

根据题意，g_i 与 g_j 满足如下关系：

$$\begin{cases} g_i - 1 = g_j, & \text{if } k_j = 0 \\ g_i - \lceil \frac{g_i}{20} \rceil = g_j, & \text{if } k_j = 1 \end{cases} \tag{6}$$

从而得到最小货物量 g 的递推关系式：

$$g_i = \begin{cases} g_j + 1, & \text{if } k_j = 0 \\ g_j + \lceil \frac{g_j}{19} \rceil, & \text{if } k_j = 1 \end{cases} \tag{7}$$

从而执行单源最短路算法即可。又由于本题的图是有向无环的，当一个结点的所有后继结点的最小货物量都求出时，即可得出该结点的最小货物量，且不存在环路依赖，因此，可以特殊化为拓扑排序算法。

Algorithm 4: Connect the POWER!

```
Input: n, V[1..n], m, E[1..m]
Output: toBuild[1..], minCost
Var toBuild be an array with dynamic length;
Var connectTo[1..n] be an array with all its elements initialized with null;
Function BFSfrom(root) begin
    Var visited[1..n] be an array with all its elements initialized with false;
    Var queue ← new Queue(root);
    visited[root] ← true;
    while ¬queue.empty() do
        u ← queue.front();
        queue.popFront();
        connectTo[u] ← root;
        forall v where (u, v) ∈ E do
            if ¬visited[v] then
                queue.pushTail(v);
                visited[v] ← true;
            end
        end
    end
end
minCost ← 0;
forall u ∈ V do
    if connectTo[u] == null then
        Call BFSfrom(u);
        toBuild.append(u);
        minCost ← minCost + 1000e4;
    end
end
```

算法描述

如算法5。

Algorithm 5: Cargo Transportation

Input: $V, E, S, T, m, k[1..n]$
Output: $minAmount$
Var $degree[1..|V|]$ be an array of out-degree of each vertex with each element initialized with 0;
Var $distance[1..|V|]$ be an array with each element initialized with $+\infty$;
forall $(u, v) \in E$ **do**
    $degree[u] \leftarrow degree[u] + 1$;
end
$distance[T] \leftarrow m$;
Var $queue \leftarrow$ new $Queue(T)$;
`// use Topological Sorting to find the shortest path from` T ;
while $\neg queue.empty()$ **do**
    $v \leftarrow queue.\text{popFront}()$;
    forall u *where* $(u, v) \in E$ **do**
        $cost \leftarrow \begin{cases} 1, & \text{if } k[v] == 0 \\ \lceil \frac{distance[v]}{19} \rceil, & \text{if } k[v] == 1 \end{cases}$;
        $distance[u] \leftarrow \min\{distance[u], cost + distance[v]\}$;
        $degree[u] \leftarrow degree[u] - 1$;
        if $degree[u] == 0$ **then**
            $queue.\text{pushTail}(u)$;
        end
    end
end
$minAmount \leftarrow distance[S]$;

时间复杂度分析

进行了一次拓扑排序算法：由于是有向无环图，$queue.\text{pushTail}$ 和 $queue.\text{popFront}$ 执行了最多 $|V|$ 次；松弛操作执行了最多 $|E|$ 次。初始化的时间复杂度为 $\Theta(|E|)$，其余语句的复杂度均为常数。因此整个算法的时间复杂度为 $O(|E| + |V|)$。