Greedy Algorithm: Please must sort the data before algorithm

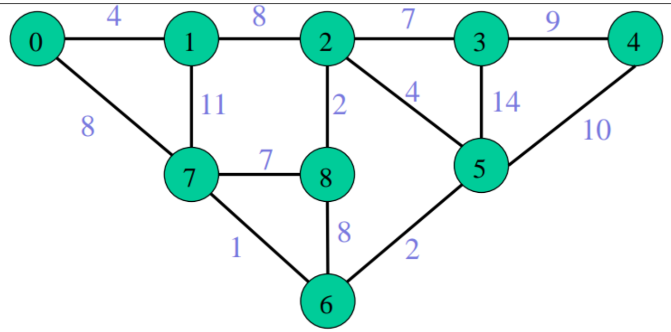
	Item	PQ	
	Dijkstra 即从一点到所有点的最短距离	Yes	**start** to all calculate the shortest path **Running time： V+E**
	Huffman prefix tree	Yes	Full tree + left branch is 0 + right branch is 1 Alphabet has 27 characters(26 letters and space) **Running time： nlogn**
Mini Spanning Tree	Prim	Yes	**Select the node randomly** Choose the shortest edge every time Running time:
	Kruskal Union Find	Yes	Select the **shortest edge firstly，整体逻辑首先对边进行Sort，每次运行选择权重最小的边。** **Running time： ElogE+E**
	Activity Selection		Sort按照结束和开始时间都可以 **重点**识别最大的子序列，即最多能安排课程（并不代表时间和收益最大） **Running time： nlogn**
	Interval Scheduling		Sort按照结束和开始时间都可以 重点是最小的教室数量安排所有的课程 **Running time： nlogn**

Dijkstra

	Step	Code: s-start node; l- edge weight	time
	PQ	Dij(G, s, l): Initialize PQ	

	Define infinite for all vertex	For v in V
		  dist(v) == infinite
		  Q.insert(v)
		dist(s) = 0
			V

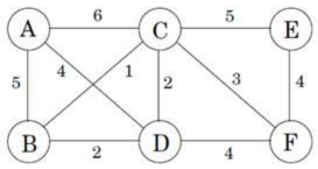
	Find the shortest path	While Q not empty: v = Q.pop() For (v, w) in G: If dist(v) + l(v,w) < dist(w): dist(w) = dist(v) + l(v,w) Q.append(w)	V+E

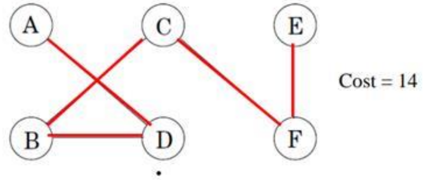


Iter	PQ (non-inf) nodes	d(0)	d(1)	d(2)	d(3)	d(4)	d(5)	d(6)	d(7)	d(8)
0	[0]	0	∞	∞	∞	∞	∞	∞	∞	∞
1	[1, 7]	0	4	∞	∞	∞	∞	∞	8	∞
2	[7, 2]	0	4	12	∞	∞	∞	∞	8	∞
3	[6, 2, 8]	0	4	12	∞	∞	∞	9	8	15
4	[5, 2, 8]	0	4	12	∞	∞	11	9	8	15
5	[2, 8, 4, 3]	0	4	12	25	21	11	9	8	15
6	[8, 3, 4]	0	4	12	19	21	11	9	8	14
7	[3, 4]	0	4	12	19	21	11	9	8	14
8	[4]	0	4	12	19	21	11	9	8	14
9	[]	0	4	12	19	21	11	9	8	14

Prim

	Step	Code: s-start node; l- edge weight	time
	初始化将所有点为无穷大	Prim(G, w): For v in V: cost(v) = infinite Prev(v) = nil	红色字体: VlogV
	任意选择一个点作为起点	pick any initial node as u cost(u) = 0 PQ.append(u)	
	运用PQ逻辑每次对相邻最小的进行弹出	While PQ: v = PQ.pop()	
	将新发现的最小边的点加到PQ	For each {u, z} in G[v]: If cost[z] > w(u,z) cost[z] = w(u,z) Prev(z) = u PQ.append(z)	蓝色字体: ElogE

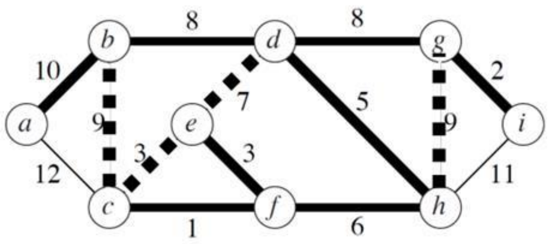




Set S	A	B	C	D	E	F
$\{\}$	0/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
A		$5/A$	$6/A$	$\boxed{4/A}$	∞/nil	∞/nil
A, D		$\boxed{2/D}$	$2/D$		∞/nil	$4/D$
A, D, B			$\boxed{1/B}$		∞/nil	$4/D$
A, D, B, C					$5/C$	$\boxed{3/C}$
A, D, B, C, F					$\boxed{4/F}$	

Kruskal

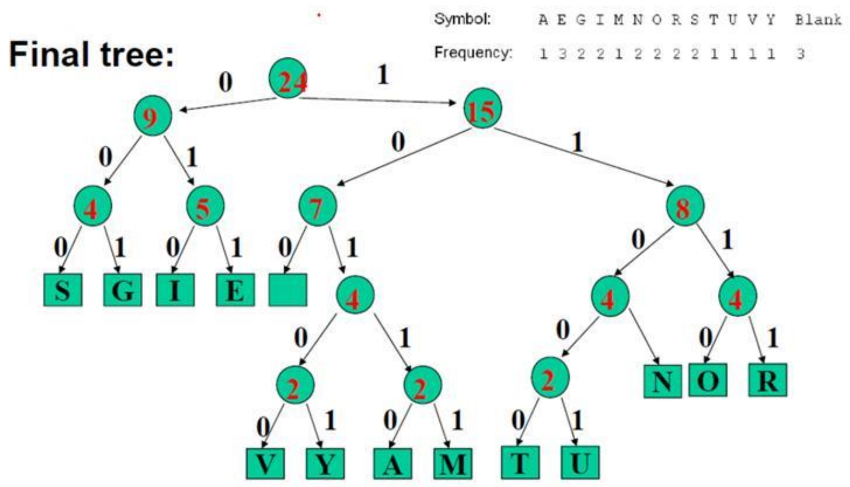
<table>
<tr><th></th><th>Step</th><th>Code: s-start node; l- edge weight</th><th>time</th></tr>
<tr><td></td><td>提取每个边的权重，然后增加到PQ</td><td rowspan="5">MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
create a single list of the edges in G.E
sort the list of edges into monotonically increasing order by weight w
for each edge (u, v) taken from the sorted list in order
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A</td><td rowspan="3">第一部分时间复杂度最高为ElogE</td></tr>
<tr><td></td><td>从最小权重开始Loop</td></tr>
<tr><td></td><td>Union Find判断它们是否相同</td></tr>
<tr><td></td><td>如果不相同将它们添加</td><td rowspan="2">第二部分就是边的比较
E</td></tr>
<tr><td></td><td></td></tr>
</table>



(c, f) : safe
(g, i) : safe
(e, f) : safe
(c, e) : reject
(d, h) : safe
(f, h) : safe
(e, d) : reject
(b, d) : safe
(d, g) : safe
(b, c) : reject
(g, h) : reject
(a, b) : safe

Huffman

	Step	Code:	time
Place the elements into PQ	1.Find to root nodes with **smallest** value 2.Create a new root node with two small nodes as children	Human(C, prob) For x in C: PQ.append(x) For i = 1 to len(C) -1: x = PQ.pop() y = PQ.pop() z = x + y PQ.append(z) Return PQ.pop()	第一部分由于x 有n个，每次PQ为logn nlogn
Remove the first two elements			nlogn
Combine 2 elements with root			
Insert the new element into PQ			



E.g., code word for “Y” is “10101”

Sum of internal node values = total weighted pathlength of tree = $\Sigma W_i \cdot L_i$
= 4+5+9+2+2+4+7+2+4+4+8+15+24 = 90 (vs. $\Sigma W_i \cdot L_i$ = 120 in naïve 5 bit per symbol code)

考点:
1.对应字母的code word
2.Total Weight
3. 画出这个图

Activity Selection

Maximizing Rental Income

```python
def recursive_activity_selector(s, f, r, k, n):
    # s: start times of activities
    # f: finish times of activities
    # r: rental incomes of activities
    # k: index of last selected activity
    # n: number of activities
    m = k + 1
    best_value = 0
    best_activities = []

    while m <= n and s[m] < f[k]:  # check compatibility
        m += 1

    if m <= n:
        # Calculate the value for selecting the current activity
        value, activities = recursive_activity_selector(s, f, r, m, n)

        if value + r[m] > best_value:
            best_value = value + r[m]
            best_activities = activities + [m]

    return best_value, best_activities
```

Maximizing longest time

```python
def recursive_activity_selector_time(s, f, k, n):
    # s: start times of activities
    # f: finish times of activities
    # k: index of last selected activity
    # n: number of activities
    m = k + 1
    best_time = 0
    best_activities = []

    while m <= n and s[m] < f[k]:  # check compatibility
        m += 1

    if m <= n:
        # Calculate the time for selecting the current activity
        total_time, activities = recursive_activity_selector_time(s, f, m, n)

        time = (f[m] - s[m]) + total_time  # Add duration of current activity

        if time > best_time:
            best_time = time
            best_activities = activities + [m]

    return best_time, best_activities
```

最大的子序列

```python
def recursive_activity_selector(s, f, k, n):
    m = k + 1
    # Find the first activity compatible with the activity at index k
    while m <= n and s[m] < f[k]:
        m += 1

    # If there's a compatible activity
    if m <= n:
        # Select activity am and recursively find compatible activities
        return [m] + recursive_activity_selector(s, f, m, n)
    else:
        # Return an empty list if no compatible activity
        return []
```

当前Code延伸可以实现最大收益以及最大时间

Interval Scheduling

```python
def allocate_classrooms_by_end(lectures):
    # Sort lectures by end times (second element of each tuple)
    lectures.sort(key=lambda x: x[1])  # Sorting by end time

    classrooms = []  # Each classroom is represented by a list of
    num_classrooms = 0

    for lecture in lectures:
        start, end = lecture  # start time and end time of the cu
        allocated = False

        # Check if the lecture can fit into any existing classroo
        for i in range(num_classrooms):
            if classrooms[i][-1] <= start:  # If the last lecture
                classrooms[i].append(end)  # Add this lecture's end time to the classroom
                allocated = True
                break

        # If no existing classroom can accommodate the lecture, allocate a new classroom
        if not allocated:
            classrooms.append([end])  # Create a new classroom and add the lecture's end
            num_classrooms += 1  # Increment the classroom count

    return num_classrooms, classrooms
```

1. **Sorting the lectures by their end times**:
   - We need to sort the list of `n` lectures based on their finish times.
   - The time complexity for sorting a list of size `n` is **O(n log n)** using algorithms like quicksort or mergesort.
2. **Allocating lectures to classrooms**:
   - After sorting, we iterate over each lecture and attempt to assign it to a classroom.
   - For each lecture, we check if it can fit into any of the previously allocated classrooms.
   - In the worst case, if there are `k` classrooms and the lecture doesn't fit into any of them, a new classroom will be allocated.
   - So, for each lecture, we perform at most `k` comparisons (one for each classroom).
   - In the worst case, the number of classrooms required will be equal to `n` (if no lectures can be scheduled in the same room), leading to a worst-case time complexity of **O(n * k)**, where `k` is the number of classrooms.

教室	End time
1	
2	
3	

首先比较第一个教室的结束时间，逐渐递增比较