CSCI 570(Victor)

Recorder:Jiechang Shi

2018 年 11 月 18 日

1 Lecture 1-Aug. 23

1. Runtime Complexities

- Worst, Best or Avg Case
- $f(n) = O(g(n))$ if g(n) eventually dominates f(n), Upper Bound
- 什么是 Efficient Algorithm? Polynomial!
- $f(n) = \Omega(g(n))$ if f(n) eventually dominates g(n), Lower Bound
- $f(n) = \Theta(g(n))$, both
- 作业错题：
  3. Suppose that f(n) and g(n) are two positive non-decreasing functions such that f(n) = O(g(n)). Is it true that log f(n) = O(log g(n))?
- True，要注意的是 $O(g(n))$ 仅对 $g(n) > 0$ 有定义，所以本题中 $log(g(n)) > 0$，$g(n) > 1$
- Discussion 错题 1：
  What is the Big-O runtime complexity of the following functions?

```
void bigOh1(int[] L, int n)
        while (n > 0)
                find_max(L, n); //finds the max in L[0···n-1]
                n = n/4;
```

- 答案：$O(n)$

- Discussion 错题 2：
  What is the Big-O runtime complexity of the following functions?

```
string bigOh2(int n)
        if(n == 0) return "a";
        string str = bigOh2(n-1);
        return str + str;
```

- 答案：$O(2^n)$，需要注意字符串合并无法在 O(1) 的时间内完成

2. Graphs

- Default: All Graph are simple Graph. 无多重边、无自环
- Directed, Undirected, Weighted, Path, Simple Path, Cycle, Acycle, Connected, Disconnected, Degree
- Handshaking Theorem:

$$2E = \sum_{x \in V} deg(x)$$

- Handshaking Theorem In directed graph:

$$E = \sum_{xinV} indeg(x) = \sum_{xinV} outdeg(x)$$

- Adjacency List or Adjancency Matrix
- Spare and Dense Graph
- Tree (acyclic and connected), 等价证明
  (a) G is a tree
  (b) Every two nodes of G are joined by a unique path
  (c) G is connected and V=E+1
  (d) G is acyclic and V=E+1
  (e) G is acyclic and if any two nod-adjacent nodes are joined by an edge, the resulting graph has exactly one cycle
- 证明 (d)->(e)
  – Using Induction

  - Base case: $|V| = 2$ $|E| = 1$ G is acyclic and $|E| = 2$ G has one cycle
  - Assume: $|V| = k$ & $|E| = k-1$ G is acyclic and add one edge G has one cycle
  - $|V| = k+1$: $|E| = k$ G is acyclic
  - Add one edge for 2 cases:
  - cases 1: the edge is between two node |V-1|, based on assume G has one cycle
  - cases 2: the edge is between the new vertice and one vertice in |V-1|, there is one and the only cycle between these two vertices
- 证明 (e)->(a)
  - Proof by contradiction
  - if G is disconnected which means there is not a path between vertice u and v
  - then add a edge between u and v
  - the edge is the only path between u and v and not have cycle, which is counter to (e)
- Corollary: Every nontrivial tree has at least two vertices of degree 1

  Proof by using Handshaking Theroem
- height of the tree: 从 0 开始计算，只有根节点的树，height=0
- 作业错题：

  1. A binary tree is a rooted tree in which each node has at most two children. Show by induction that in any nonempty binary tree the number of nodes with two children is exactly one less than the number of leaves.
- 注意：需要使用 induction 证明

3. Graph Traversals(DFS & BFS)
   - Runtime complexity: $O(V + E)$ which is **linear** with the input size

- Topological Sort for DAG
  - 正向寻找，非线性时间
  - 逆向寻找，线性时间
    从**每一个**入度为 0 的点开始 DFS，离开节点时输出
- Strongly Connected Graphs
  - Brute Force: $O(V(V+E))$
  - Better Algorithm:$O(V+E)$
    DFS on the same vertice in G and G^T and return whether all vertices is reachable
    Proof:
    In G, means the vertice can reach every vertice
    In G^T, means every vertice can reach the vertice

4. Planar Graphs(平面图)
   - **Euler's Formula**: V-E+F=2
     proof by induction based on edge
   - 4 Color Theorem
   - Proof of 6 Color by induction based on vertices
     lema: Any planar graph contains a vertex of degree less than 6
   - 如果非平面图的填色问题，是否有线性时间的解决方案？maybe
5. Bipartite Graph(二分图)
   - 如何将问题转化为二分图问题

2 Lecture 2-Aug. 30

1. Topic: Heaps & Amortized Analysis

2.1 Heaps

1. Complete Binary Tree
2. Binary Heaps

- Default: min-heap
- Invariants:
  Structure Property
  Ordering Property
- Operations: insert & deleteMin & decreaseKey & build & merge
- Implementation: simple array
- Insertion:$O(logn)$
- deleteMin:$O(logn)$
  Merge Sort 和 Heap Sort 哪个快?
  Heap Sort 在最坏情况下 $O(nlogn)$, Merge Sort 总是 $O(nlogn)$
- decreaseKey:$O(logn)$
  为什么不考虑 increaseKey?
  increase 总是不会影响到我们关注的最小值
- Build a Heap:$O(n)$
  - by insertion: $O(nlogn)$
  - heapify: $O(n)$
  - Starting at position $n/2$ and working toward position 1, push down
  - proof the complexity

$$\sum_{k=1}^{h} k * 2^{h-k} = 2^h * \sum_{k=1}^{h} \frac{k}{2^k} \leq 2^h * \sum_{k=1}^{\infty} \frac{k}{2^k}$$

$$x = 2^h * \sum_{k=1}^{\infty} \frac{k}{2^k} = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \dots$$

$$\frac{x}{2} = \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \dots$$

$$x - \frac{x}{2} = \frac{1}{2} + \frac{2-1}{4} + \frac{3-2}{8} + \dots = \sum_{k=1}^{\infty} \frac{1}{2^k} = 1$$

$$\sum_{k=1}^{h} k * 2^{h-k} \leq 2^h * 2 = 2^{h+1} = O(n)$$

3. Binomial Heaps

- 希望改进 deleteMin 或者 decreaseKey
- deleteMin 无法改进，因为基于比较的排序算法最坏情况最优为 $O(nlogn)$
- 改进 decreaseKey!
- 循环定义 Binomial Tree
  - B_0 is a single node
  - B_k is formed by joining two B_{k-1} trees
- 为什么叫 Binomial Tree(组合数树)?
  因为 rank=n 的 Binomial Tree 的第 k 层有 $\binom{k}{n}$ 个节点
  rank=n 的 Binomial Tree 有 $\sum \binom{k}{n} = 2^n$ 个节点
- Binomial Heap 的定义
  - pointer for minimun and a linklist for all binomial tree roots
  - In a binomial heap there is **at most** one binomial tree of **any** given rank
- 当要存储 n 个元素时，需要哪些 rank 的 Binomial Tree?
  将 n 转换为二进制，若第 i 位为 1，则需要 rank=i 的 Binomial Tree
- Merging two Heaps:
  - Binary Heap: Heapify $O(n)$
  - Binomial Heap:
    merge two linklist $O(1)$
    merge if two tree have the same ranks $O(logn)$
    Total:$O(logn)$
- deleteMin in Binomial Heap:
  Remove the root of B_k $O(1)$
  Got k sub binomial tree to the heap $O(logn)$
  Merge $O(logn)$
  Total: $O(logn)$
- Insertion in Binomial Heap:
  Merge a B_0 in worst cast $O(logn)$
  In Amortized Analysis: $O(1)$ Just like **binary counter problem**

4. Fibonacci Heaps

   - Idea: **relaxed** binomial heaps
   - Goal: decreaseKey in $O(1)$
   - 将顶端列表的调整全部放在 deleteMin 中完成

5. Compare Heaps

	Binary	Binomial	Fibonacci
findMin	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
deleteMin	$\Theta(logn)$	$\Theta(logn)$	$O(logn)$(ac)
insert	$\Theta(logn)$	$\Theta(1)$(ac)	$\Theta(1)$
decreaseKey	$\Theta(logn)$	$\Theta(logn)$	$\Theta(1)$(ac)
merge	$\Theta(n)$	$\Theta(logn)$	$\Theta(1)$(ac)

2.2 Amortized Analysis

1. Amortized analysis gives **the average performance** of each operation **in the worst case**.

2. Unbounded Array(Aggregate analysis)

   - create a array of a fixed size, if the array is full create a array of double size and copy the former array
   - insertion: best case $O(1)$, worst case $O(n)$
   - Average?
     For $n = 2^h + 1$ insertion:
     insert $2^h + 1$ time
     copy $1 + 2 + 4 + 8 + \cdots + 2^h = 2^{h+1} - 1$ time
     total $2^h + 2^{h+1} = 3n = O(n)$ time
     For each insertion: $\frac{3n}{n} = O(1)$ time

3. Binary counter(Aggregate analysis)

   - Flip a binary number cost O(1), add from 000...0 to 111...1 and calculate the average cost
   - best case $O(1)$, worst case $O(logn)$

- Average?
  For each binary number, the last one flip every time, the second last one flip every two time and so on...
  Total: $n + n/2 + n/4 + n/8 + \cdots \leq n * 2 = O(n)$
  Avg:$2n/n = O(1)$
- Assume the least significant bit k=0, the most significant bit k=log n, and flip a bit cost 2^n
  Avg:$O(logn)$

4. Discussion 讲的 Token 法 (Accounting Method)

- 对于一些问题不同的操作序列会极大影响 Cost 的计算需要使用 Token 法
- 例题：
  1. You have a stack data type, and you need to implement a FIFO queue. The stack has the usual POP and PUSH operations, and the cost of each operation is 1. The FIFO has two operations: ENQUEUE and DEQUEUE. We can implement a FIFO queue using two stacks. What is the amortized cost of ENQUEUE and DEQUEUE operations.
- 算法：
  PUSH: push to stack 1
  POP: if the stack 2 is empty then pop every node from stack 1 to stack 2
  if the stack 2 is full then pop stack 2
- Amortized Analysis:
  无法使用 Aggregate Analysis，因为 cost 与操作序列相关
  引入 Accounting Method(Token 法 & credit 法)
- 我们将一个操作的 Amortize 费用中超过本身费用的部分称为 Credit 用于支付其他操作的部分费用
- Aggregate 和 Accounting 的区别：
  Aggregate 先计算总体费用再**均摊**到每一个操作上，每一个操作

均摊费用相同

Accounting 一些操作为另一些操作付费，每一种操作的费用**可能不同**

- 注意，对于任何 Amortize 分析需要满足：

$$\sum_{i=1}^{n} \hat{c_i} \geq \sum_{i=1}^{n} c_i$$

- 在 Accounting Method 中将 credit 定义为：

$$\sum_{i=1}^{n} \hat{c_i} - \sum_{i=1}^{n} c_i$$

- 由于不等式，credit 在**任何时候**均**不能为负**
- In Binary Counter Problem:
  We set flip a binary bit from 0 to 1 cost O(2):
  1 for the flip
  1 for flip it from 1 to 0 in later operation
  For each increasment, we need to pay one and the only time which take O(2)
- In stack to queue problem:
  We set push cost O(3):
  1 for push to stack 1
  1 for pop from stack 1 in later operation
  1 for push to stack 2 in later operation
  We set pop cost O(1):
  1 for pop from stack 2
  The Avg cost for push and pop is O(1)

5. 上课没讲的 Potential Method

- 使用 Potential Function$\Phi(D_i)$ 来表示第 i 操作的 Potential
- 定义 Amortized Cost 为：

$$\hat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

- 进一步的有：

$$\sum_{i=1}^{n} \hat{c_i} = \sum_{min}^{max} c_i + \Phi(D_n) - \Phi(D_0)$$

- 带入 $\sum_{i=1}^{n} \hat{c_i} \geq \sum_{i=1}^{n} c_i$ 有下式在任何时候成立:

$$\Phi(D_k) \geq \Phi(D_0)$$

  通常有 $\Phi(D_0) = 0$，则下式在任何时候成立：

$$\Phi(D_k) \geq 0$$

- In Binary Counter Problem:
  We define $\Phi(D_i)$ is the number of 1 in the binary bit of i
  In step i assum we flip t_i bits from 1 to 0, so:
  $c_i = t_i + 1$
  $\Phi(D_i) - \Phi(D_{i-1}) = 1 - t_i$
  $\hat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1}) = (t_i + 1) + (1 - t_i) = 2$

3 Greedy Algorithm

1. 为什么使用 Greedy?
   Simplicity & Efficiency

2. There is **no guarantee** that such a greedy algorithm exists, however a problem to be solved must obey the 2 properties:
   greedy-choice property
   optimal substructure

3. Prooft of greedy-choice property is usually by **contradiction**

4. Proof of optimal substruture correctness is usually by **induction**

5. 选择最多非重叠线段问题：

   - Algorith:
     Sort requests with respect to the finish time $f(i)$

Pick a request that has the earliest $f(i)$
Complexity:$O(nlogn)$

- Proof:
  Let $A = i_1, i_2, \ldots, i_k$ be our schedule
  Let $Opt = j_1, j_2, \ldots, j_m$ be the optimal schedule
  We prove by induction that $f(i_r) \leq f(j_r)$ for $\forall r \leq k$ and thus our solution cannot be worse than the optimal one
  Base case, r=1, True
  Assume, $f(i_{r-1}) \leq f(j_{r-1})$
  Note, $f(j_{r-1}) \leq s(j_r)$
  Thus, by IH, $f(i_{r-1}) \leq s(j_r)$
  Since j_r is in the set of available intervals
  Our algorithm will pick j_r, it follows $f(i_r) \leq f(j_r)$
  Then we need to prove $k \geq m$
  By contradiction. Let $k < m$:
  There must be a request j_{k+1}, s.t. $f(j_k) \leq s(j_{k+1})$
  But $f(i_k) \leq f(j_k)$
  It follows, $f(i_k) \leq s(j_{k+1})$
  j_{k+1} does not overlap any in our schedule
  So our algorithm would not stop at i_k and choose j_{k+1} as the next request. Contradiction

6. The Minimum Spanning Tree

   - The number of spanning trees in K_n is n^{n-2}
   - Kruskal's Algorithm
     Complexity:
     Sorting $O(ElogE)$
     Cycle detection $O(V)$
     Total: $O(V * E + ElogE)$
   - Prim's Algorithm
     Complexity by simple array:
     $O(V^2 + E)$
     Complexity by binary heap:

$O(VlogV + ElogV)$
Complexity by Fibonacci heap:
$O(VlogV + E)$
需要考虑到图的稀疏性选择使用 array 或 heap

- Proof of Prim Algorithm
  Lemma: Given any cut in a graph, the crossing edge of minimum weight is in the MST of the graph
- 上课例题：
  If a connected undirected graph G=(V,E) has V + 1 edges, we can find an MST of G in O(V) runtime.
  True, 对于图 G 的每一个 Cut 至多只有 3 个 neiborhood，否则不连通

7. Shortest Path Problem

   - Dijkstra's Algorithm
   - Complexity by binary heap:
     INIT: $O(V)$
     LOOP:
     Deletemin For each Vertices $O(VlogV)$
     DecreaseKey For each Edges $O(ElogV)$
     TOTAL: $O(VlogV + ElogV)$
   - Complexity by array:$O(V^2 + E)$
   - **Dijkstra 不适用于负权边！**
   - Design a **linear time** algorithm to find Shortest Path in **DAG**
     Find a Topological order
     From the first vertices to update the vertices connect to it
     Complexity: $O(V + E)$

8. 作业错题：
   1. You are given a set X = {x_1, x_2, ⋯, x_n} of points on the real line. Your task is to design a greedy algorithm that finds a smallest set of intervals, each of length-2 that contains all the given points. Linked list is a data structure consisting of

a group of nodes which together represent a sequence. Under the simplest form, each node is composed of data and a reference (in other words, a link) to the next node in the sequence.
a) Describe the steps of your greedy algorithm in plain English. What is its runtime complexity?
b) Argue that your algorithm correctly finds the smallest set of intervals.

- part A:

```
Sort X = {x1,x2, ··· ,xn}. Let it be S
Initial T={}
while S not empty:
        select the smallest x from S
        add[x,x+2] into T
        remove all elements within [x,x+2] from S
return T
```

- part B: Need to **proof by contradiction and induction**!
  Assume there is a better solution OPT include $\{j_1,j_2,...,j_m\}$
  Our solution T include $\{i_1,i_2,...,i_n\}$
  First we need to prove $Left(i_k) \geq Left(j_k) \forall k \in [1,m]$by induction
  Base case,k=1,$Left(i_1)$ is the first point in S, if $Left(j_1) > Left(i_1)$ then the first point is not cover in OPT
  Inductive Step: assume it is true for all $Left(i_r) \geq Left(j_r)$, we will proof $Left(i_{r+1}) \geq Left(j_{r+1})$
  Note $Right(i_r) \geq Right(j_r)$ when $Left(i_r) \geq Left(j_r)$
  Assume x_p is the leftmost point cover by i_{r+1}, so $x_p > Right(i_r) \geq Right(j_r)$
  So $Left(j_{r+1}) \leq x_p = Left(i_{r+1})$
  Now we proofed $Left(i_k) \geq Left(j_k) \forall k \in [1,m]$ then we need to proof $n \leq m$

By contradiction, we assume $n > m$

For i_{m+1}, we picked this intervals due to $x_r = Left(i_{m+1}) > Right(i_m)$ cannot cover by interval i_m

Due to the Lemma, point x_r cannot cover by any interval in OPT solution,contradiction

4 Divide-And-Conquer Algorithms

1. Binary Search & MergeSort
2. Recurrence Equation
   $T(n) = a * T(n/b) + f(n)$
3. Counting leaves:
   height: $log_b n$
   leaves: $a^{log_b n} = a^{log_b a * log_a n} = n^{log_b a}$
4. Master Theorem:
   Let $c = log_b a$
   Case 1(only leaves): $f(n) = O(n^{c-\epsilon})$ then $T(n) = \Theta(n^c)$ for some $\epsilon > 0$
   Case 2(all nodes): $f(n) = \Theta(n^c log^k n)$,$k \geq 0$,then$T(n) = \Theta(n^c log^{k+1} n)$
   Case 3(only internal nodes): $f(n) = \Omega(n^{c+\epsilon})$ then $T(n) = \Theta(f(n))$ for some $\epsilon > 0$
5. 课件没写的 Case 3 的额外条件：$af(n/b) \leq cf(n)$ for some constant c<1
6. 需要注意：
   f(n)>0 & a>1 & b>0 & ϵ>0 & k>0
7. Proof of Case 1:

$$T(n) = T(1) * n^{log_b a} + \sum_{k=0}^{log_b n - 1} a^k f(\frac{n}{b^k})$$

$$f(n) = O(n^{log_b a - \epsilon})$$

$$\sum_{k=0}^{log_b n - 1} a^k f(\frac{n}{b^k}) \leq c \sum_{k=0}^{log_b n - 1} a^k (\frac{n}{b^k})^{log_b a - \epsilon} = cn^{log_b a - \epsilon} \sum_{k=0}^{log_b n - 1} (\frac{a}{b^{log_b a}})^k b^{\epsilon k}$$

$$\sum_{k=0}^{log_b n-1}(\frac{a}{b^{log_b a}})^k b^{\epsilon k} = \sum_{k=0}^{log_b n-1} b^{\epsilon k} \leq \sum_{k=0}^{\infty} b^{\epsilon k} = \frac{1}{1-b^{\epsilon k}}$$

So:

$$T(n) = n^{log_b a} + \frac{c}{1-b^{\epsilon k}} n^{log_b a-\epsilon} = n^{log_b a} + c_0 n^{log_b a-\epsilon} = \theta(n^{log_b a})$$

8. Proof of Case 2(k=0):

$$T(n) = T(1) * n^{log_b a} + \sum_{k=0}^{log_b n-1} a^k f(\frac{n}{b^k})$$

$$f(n) = \Theta(n^{log_b a}) = cn^{log_b a}$$

$$\sum_{k=0}^{log_b n-1} a^k f(\frac{n}{b^k}) = c \sum_{k=0}^{log_b n-1} a^k (\frac{n}{b^k})^{log_b a} = cn^{log_b a} \sum_{k=0}^{log_b n-1} (\frac{a}{b^{log_b a}})^k = cn^{log_b a} log_b n = \Theta(n^{log_b a} logn)$$

$$T(n) = \Theta(n^{log_b a}) + \Theta(n^{log_b a} logn) = \Theta(n^{log_b a} logn)$$

9. Integer Multiplication:
   将数分为两个部分，x1,x0,y1,y0
   x1y0+x0y1=(x1+x0)(y1+y0)-x1y1-x0y0
   只需要计算 3 个乘法
   $T(n) = 3T(n/2) + O(n)$
   $T(n) = n^{log_2 3}$
   进一步如果分成 3 个部分可以更快
   进一步 Fast Fourier Transform:$O(nlognlog(logn))$

10. Matrix Multiplication
    Strassen's Algorithm

11. Finding the Maximum Subsequuence Sum
    Find Max on the left and Max on the right
    Find the span
    $T(n) = 2T(n/2) + O(n)$

12. Closest Pair of Points
    Computer separation line L O(nlogn)
    The left part T(n/2)

The right part T(n/2)
Find all points in 2δ zones O(n)
Sort by y-axis O(nlogn)
For each point in zone find it's 7 neiborhood O(n)
Total T(n)=2T(n/2)+O(nlogn)=$O(n(logn)^2)$

13. Discussion 错题 1:
    3. There are 2 sorted arrays A and B of size n each. Design a D&C algorithm to find the median of the array obtained after merging the above 2 arrays (i.e. array of length 2n). Discuss its runtime complexity.
    Step 1:Find the median of A, m_a and median of B, m_b
    Step 2:If $m_a < m_b$ then delete all element less than m_a in A and all element large than m_b in B
    Step 3:Go back to Step 1 until A and B only have 2 number
    Complexity: O(log n)

14. Discussion 错题 2：
    5. You are given an unsorted list of ALL integers in the range [0,..., $2^k - 1$] except for one integer, denoted the missing number by M. Describe a divide-and-conquer to find the missing number M, and discuss its the worst-case runtime complexity in terms of n = 2^k.
    按二进制位分治
    Worst case:O(n)

15. 上课没讲的猜测法：
    Making a good guess
    Subtleties(做更大的放松)
    Avoiding pitfalls(需要严格证明到 Guess)
    Changing variables

5 Dynamic Programming

1. Optimal Substructure & Overlapping subproblem

2. Fibonacci Number
   Time Complexity:$O(n^2)$
   需要注意：数的加法需要 $O(n)$ 复杂度

3. Memoization & Tabulation
   Default: In this class, DP is Tabulation

4. The Money Changing Prolbem
   Time Complexity:$O(nk)$

5. 0-1 knapsack Problem
   Time Complexity:$O(nw)$

6. Draw the Decision Tree to Find the overlapping subproblem

7. Longest Common Subsequence
   Why not Longest Common Substring?
   Time Complexity:$O(nm)$

8. 需要注意如何从动规结果表格中获得路径

9. Static Optimal Binary Search Tree
   为什么不用平衡树？
   In balanced Tree, each node have **the same possibility** to arrive.

10. 来自作业：解决 DP 问题的四个步骤

    - Define (in plain English) subproblems to be solved.
    - Write the recurrence relation for subproblems.
    - Write pseudo-code to compute the optimal value
    - Compute the runtime of the above DP algorithm in terms of n.

11. 来自 Discussion：Recurrence Relation Equation 相关:
    需要考虑初始条件，且初始条件尽量确保右端为常数

12. 来自 Discussion：为什么用 DP？
    将 Exponential 的问题转换为 Polynomial
    But DP cannot solve all problems in real polynomial time!

13. Bellman-Ford Algorithm
    $D[v,k]$ denotes the length of the shortest path from s to v that uses at most k edges
    How to find out if a graph has negative cycle?
    Do one more time!

14. Floyd-Warshall Algorithm
    D[i,j,k] denotes the shortest path from i to j which all intermediate vertices can *only* by chosen from the set {1,2,...,k}
    How to find out if a graph has negative cycle?
    Check the main diagonal
    How to get the path?
    Record k for each update

15. Compares for All-pairs problem:
    Dijkstra: $O(V(E+VlogV))$, 如果没有负权边最快选项
    Floyd-Warshall: $O(V^3)$
    Bellman-Ford: $O(EV^2)$

16. Chain Matrix Multiplication:
    思考题：Brute Force 的复杂度
    $O(4^n)$，不是 $O(n!)$!
    DP Complexity:$O(n^2)$
    How to recover the optimal set of parentheses?

17. Polygon Traiangulation Problem

6 Midterm 错题整理

6.1 Q3

Let us consider California coast with its many beautiful islands. We can picture the coast as a straight line: the mainland is on one side while the sea with islands is on the other side. Despite the remote setting, the residents of these islands are avid cell phone users. You need to place cell phone base stations at certain points along the coast so that every islands

is within D miles of one of the base stations. Give an efficient algorithm that achieves this goal and uses as few base stations as possible. You may assume that each island has xy-coordinates (x_1, y_1), (x_2, y_2), $\cdots$, (x_n, y_n).

6.1.1 PartA

Design a greedy algorithm to find the minimum number of stations required to cover all islands.

Step 1: Draw a cycle of radium D with each island as center. Caculate the left $L_i = x_i - \sqrt{D^2 - {y_i}^2}$ and right $R_i = x_i + \sqrt{D^2 - {y_i}^2}$ of intersection of the x-axis, i=1,2,...,n

Step 2: Sort the (L_i, R_i) by R_i, 类似于任务覆盖问题应该依照结束点排序

Step 3: Starting from the smallest R_i, install the radar at R_i and delete all intervals that overlaps with it. To delete we need to find R_j such that $L_j > R_i$ but $L_{j-1} < R_i$4

Step 4: Repeat until all intervals are deleted

6.1.2 PartC

Prove the correctness of your algorithm.

证明完全相当于任务覆盖问题

Let $(I_1, I_2 \ldots, I_m)$ be our solution and $(J_1, J_2 \ldots, J_p)$ the optimal solution.

Consider the first stations I_1. The optimal solution J_1 cannot be on the left of our stations because its cover area will be less than D. Also J_1 cannot be on the right of I_1 because the first island will not be covered. Thus, $I_1 = J_1$.

We can apply the same argument to all other stations to prove that our solution size is equal to the optimal size.

6.2 Q5

Given a graph G = (V, E) where a set of cities V is connected by a network of roads E. Each road/edge has a positive weight, w(u, v) between cities u and v. There is a proposal to add a new road to the network. The

proposal suggests a list C of candidate pairs of cities between which the new road may be built. Your task is to choose the road that would result in the maximum decrease in the driving distance between given city s and city t. Design an efficient algorithm for solving this problem, and prove its complexity in V, E and C.

6.2.1 Algorithm

Step 1:Use dijkstra to caculate the shortest path from s to every other point **这里使用 dijkstra 效率最高**

Step 2:Use dijkstra to caculate the shortest path from t to every other point

Step 3:For every u,v, the shortest path distance between s and t which covers road u $\rightarrow$ v is min(dist(s,u) + dist(t,v) + length(u,v),dist(s,v) + dist(t,u) + length(u,v)). **因为本题为无向图，这里需要考虑 u->v 和 v->u，所以可以 s->u->v->t 也可以 s->v->u->t**

Step 4: Choose the shortest distances from Step 3.

6.3 Runtime Complexity

Step 1:$O(ElogV)$
Step 2:$O(ElogV)$
Step 3:$O(C)$
Total:$O(C + ElogV)$

7 Network Flow

7.1 Lecture 1

1. Definition of Flow Problem:
   Given a **connected, directed, weighted** graph G=(V,E) with two distinguished vertices, s and t.
   Edge capacities $c(e)$ are integers and $c(e) \geq 0$
   We will call $f(e)$ a flow through e such that:

   (a) $0 \leq f(e) \leq c(e)$

(b) for each $v \in V - \{s, t\}$ $\sum_{in} f(e) = \sum_{out} f(e)$

2. Max-flow problem:
   Given a flow network G, find a flow f from s to t of the maximum possible value.

3. **Why Greedy doesn't work?**
   因为 Greedy 在原始图上不能做反悔

4. Residual Graph G_f:
   Forward edges: $\forall e \in E$, $f(e) < c(e)$ then $c_f(e) = c(e) - f(e)$
   Backward edges: $\forall e \in E$, $f(e) > 0$ then $c_f(e^R) = f(e)$,e^R 是 e 的反向边

5. Augment Path:
   Let P be an s-t path in G_f
   If the $bottleneck(P, f) > 0$ then we can increase the flow by sending $bottleneck(P, f)$ along the path P

6. **Ford-Fulkerson** Algorithm for Max-Flow Problem:
   (a) Start with |f|=0, so f(e)=0
   (b) Find an augmenting path in G_f by DFS
   (c) Augment flow along this path
   (d) Repeat until there is no an s-t augmenting path in G_f

7. Proof Lemma1: After augmenting we still have a flow:
   If e is forward edge
   $f'(e) = f(e) + bottleneck(P, f) \leq f(e) + (c(e) - f(e)) = c(e)$
   The new flow will never exceed the capacity.
   If e is backward edge
   $f'(e) = f(e) - bottleneck(P, f) \geq f(e) - (f(e)) = 0$
   The new flow will never have negative edge

8. Cut: A cut is a partition (A,B) of the vertices V
   The capacity of (A,B) is the sum of capacity of edges **from A to B**
   $cap(A, B) = \sum_{out} c(e)$
   Cut Capacity only count the output edge

9. Proof Lemma2: For any flow and cut: $|f| = \sum_{out} f(e) - \sum_{in} f(e)$
   Note: $\sum_{in} f(s) = 0$
   Also: $\sum_{in} f(e) = \sum_{out} f(e)$ for other node
   So: $|f| = \sum_{out} f(s) = \sum_{out} f(s) + \sum_{in} f(s) + \sum (\sum_{in} f(e) - \sum_{out} f(e))$
   So: $|f| = \sum_{out} f(e) - \sum_{in} f(e)$

10. Proof Lemma3: For any flow f and any (A,B) cut: $|f| \leq cap(A, B)$
    $|f| = \sum_{out} f(e) - \sum_{in} f(e) \leq \sum_{out} f(e)$
    Note: $f(e) < c(e)$
    $|f| \leq \sum_{out} c(e) = cap(A, B)$

11. Max-flow Theorem:
    The following conditions are equivalent for any f:

    - $\exists cut(A, B) s.t. cap(A, B) = |f|$
    - f is a max-flow
    - There is no augmenting path wrt f

12. Proof: 3 -> 1 :(其余两个是平凡的)
    How to proof ∃? Show it!
    By Lemma2:$|f| = \sum_{out} f(e) - \sum_{in} f(e)$
    Let A be a set of vertices reachable from s in G_f
    Let B be other vertices
    In Graph G:
    $\forall u \in A, \forall v \in B, f(e) = c(e)$ 否则 s 可达 v
    $\forall u \in B, \forall v \in A, f(e) = 0$ 否则 s 可达 v
    So: $|f| = \sum_{out} f(e) - \sum_{in} f(e) = \sum_{out} c(e) - \sum 0 = cap(A, B)$

13. Max-flow Min-cut Theorem:
    Value of the max-flow = capacity of the min-cut

14. Runtime Complexity of Ford-Fulkerson Algorithm: $O(|f|(E + V))$, pseudo-polynomial

15. Bipartite Match:
    Max matching = Max flow

Runtime Complexity: $Max(|f|) \leq |V|$
So Runtime is: $O((E+2V)V) = O(EV)$

16. How to improve the efficiency:

17. Using BFS instead of DFS:
    确保先处理较短的路径
    $O(V * E^2)$

18. Capacity-Scaling Algorithm:
    二分最小 Capacity 确保先处理高 bottleneck 的路径
    $O(log(|f|)E^2)$ weakly-polynomial

19. Edge Disjoint Paths:
    Every edge have a capacity of 1 and do max-flow
    Runtime Complexity $O(VE)$

20. Vertex Disgoint Paths:
    将每一个 vertex 分为两个，一个处理进入，一个处理输出，两个点之间连单向边 capacity=1
    Runtime Complexity $O(VE)$
    此法可处理对顶点赋权的最短路问题

21. From Discussion:

    - Relation between flow value and cut capacity? **Lower bound**
    - If all directed edges in a network have distinct capacities then there is a unique maximum flow.
      False, by example
    - If all edge add same value the mincut(not the value) unchange.
      False, by example
    - 错题:
      Q3: 类似于 Edge disjoint Problem，有 1 个原点，k 个终点
      新建超级终点，最大流就是答案，人员安排在最小割上
      注意：终点到超级终点的 capacity 应该为 $+\infty$ 而不是 1，**使得最小割不会包含超级终点!**

7.2 Lecture 2

7.2.1 Image Segmentation

1. Problem Definition:
   Partition an image into meaningful regions with respect to a particular application.
   V = set of pixels
   E = pairs f neighboring pixels
   $a_i \geq 0$ is likelihood pixel i in the foreground
   $b_i \geq 0$ is likelihood pixel i in the background
   $P_{ij} \geq 0$ is the penalty for placing i,j in different section
   Equivalent Representation to: Maximize

$$\sum_{foreground} a_j + \sum_{background} b_i - \sum_{(i,j)\in E} p_{ij}$$

2. Capacity of the cut:
   (s,j), where $j \in background$, contributes a_j to the cut
   (i,t), where $i \in foreground$, contributes b_i to the cut
   (i,j), where $i \in foreground \& j \in background$ contributes p_{ij} to the cut

$$cap(foreground, background) = \sum_{background} a_j + \sum_{foreground} b_i + \sum_{(i,j)\in E} p_{ij}$$

3. Equivalent Representation:

$$Q = \sum_{all} a_j + \sum_{all} b_i = constant$$

$$\sum_{foreground} a_j + \sum_{background} b_i - \sum_{(i,j)\in E} p_{ij} = Q \quad -(\sum_{background} a_j + \sum_{foreground} b_i + \sum_{(i,j)\in E} p_{ij})$$

$$= Q \qquad\qquad -cap(foreground, background)$$

4. If we want to maximize $q(foreground, background)$, we can minimize $cap(foreground, background)$, which is the **min cut** of the graph

5. Algorithm:
   Step 1:Find the max-flow

Step 2:Find the min-cut from max-flow

Step 3:Output the min-cut

7.2.2 Circulation

1. Definition:
   A set S of sources generating flow
   A set T of sinks that can absorb flow
   Goal is to send a flow from supply nodes with available supply to those with given demands

2. Circulation with Demands is $f : E- > R^+$ that satisfies:
   (Capacity) For each $e \in E$, $0 \leq f(e) \leq c(e)$
   (**Conservation**) For each $v \in V$, $f^{in}(v) - f^{out}(v) = d(v)$
   与一般网络流区别在于此处为 **d(v)** 而不是 **0**
   Necessary Conditon:
   $$\sum_{v \in V} d(v) = 0$$

3. Reduction to Max Flow:
   For each v with $d(v) < 0$, add edge (s, v) with capacity $-d(v)$
   For each v with $d(v) > 0$, add edge (v, t) with capacity $d(v)$
   There is a feasible circulation with demands d(v) in G if and only if the maximum s-t flow in G' has value D
   $$D = \sum_{d(v)>0} d(v)$$

4. Circulation with Demands and Lower Bounds is $f : E- > R^+$ that satisfies:
   (**Capacity**) For each $e \in E$, $l(e) \leq f(e) \leq c(e)$
   与前者的区别在于流有下界
   (Conservation) For each $v \in V$, $f^{in}(v) - f^{out}(v) = d(v)$

5. Reduction to Circulation with Demands:
   Define :
   $$L(v) = \sum_{etov} l(e) - \sum_{eoutv} l(e)$$

Change the demand:

$$d'(v) = f^{in}(v) - f^{out}(v) = d(v) - L(v)$$

类似于将 LowerBound 提前输入到节点中，这样会改变每一个节点的 Demand

Change of the capacity:

$$c'(e) = c(e) - l(e)$$

7.2.3 Survey Design Problem

1. Definition:
   k products, n customers
   A customer receives questions only about products he has bought
   Each customer receives number of question between $p_1(i)$ and $p_2(i)$
   Each products send number of question between $c_1(i)$ and $c_2(i)$

2. Construct of Graph:

   - Vertices(V) are n customers and k products
   - There is an edge between customer i and product j iff the customer has purchased the product,lower=0,upper=1
   - Vertex A connected to each customer with lower=$p_1(i)$, upper=$p_2(i)$
   - Each product connected to Vertex B with lower=$c_1(i)$, upper=$c_2(i)$
   - Formulate as a circulation with lower bound
     Edge from B to A, lower=$\sum p_1$, upper=$\sum p_2$
   - 为什么要添加一个 Feedback 的边：
     分两种情况讨论：
     A) If $\sum p_1 \geq \sum c_1$, We send $\sum p_1$ from A to each customers then to each product, then to B, we need to absorb the exceed flow in B which is $\sum p_1 - \sum c_1$. Add a edge with lower bound $\sum p_1$, base on **Circulation with Demands and Lower Bounds** the demand of B:

$$d(B) = f^{in}(B) - f^{out}(B) = -L(B) = \sum p_1 - \sum c_1$$

B) If $\sum p_1 < \sum c_1$, We need send $\sum c_1$ from A, but the demand of A in origin Graph is $\sum p_1 < \sum c_1$, So we need to get the exceed part from B to A which is $\sum c_1 - \sum p_1$. So B is a source in this condition. Add a edge with lower bound $\sum p_1$, base on **Circulation with Demands and Lower Bounds** the demand of B:

$$d(B) = f^{in}(B) - f^{out}(B) = -L(B) = -(\sum c_1 - \sum p_1)$$

3. Reduce to Circulation with Demands:
   $d(A) = 0$
   $d(customer) = -p_1(i)$,Source
   $d(product) = c_1(i)$,sink
   $d(B) = \sum p_1 - \sum c_1$, It depend
   $c(e) = c(e) - l(e)$

4. Claims: There is a feasible circulation with demands in G if and only if the maximum s-t flow in G' has value D:
   $D = max(\sum p_1, \sum c_1)$

5. 注意到：在最大流时，从 B 到 A 除了 LowerBound 什么都不会传输

7.2.4 Course Selection Problem

1. Graph Construction:
   G=(V,E)
   Directed edge (u,v) means v is a prerequisite to u, 注意此处为**儿子指向父亲**, c(e)=∞，保证在选 min-cut 时不会包含这些边
   s to each class with possitive value, c(e)=p(class)
   v to each class with negative value, c(e)=-p(class)

2. Flow in that graph:
   All possitive class: $\sum p(class^+) = P1$
   All negative class: $\sum p(class^-) = P2$
   Cut capacity:$cap = \sum_{unseleted} p(class^+) + \sum_{seleted} p(class^-) + k * \infty$
   Maximize: $\sum_{seleted} p(class^+) - \sum_{seleted} p(class^-) = P1 - cap$

Same as Minimize:*cap*
Same as Max-flow

8 NP Hardness

8.1 Base Knowledge

1. A problem P is **decidable (or computable)** if it can be solved by a Turing machine that halts on every input.

2. Runtime Complexity:
   Let M be a Turing machine that halts on all inputs.
   Assume we compute the running time purely as a function of the length of the input string.
   Definition: The running complexity is the function $f : N \to N$ such that $f(n)$ is the maximum number of steps that M uses on any input of length n.

3. Algorithm:
   The set Σ^* is the set of all finite sequences of elements of alphabet Σ
   A language $l \subseteq \Sigma^*$ is decidable if there is a Turing Machine M which halts on every input $x \in L$
   A problem P is decidable if it can be solved by a Turing machine T that always halts.
   We say that P has an algorithm

4. The Church-Turing Thesis:

   > Any natural/reasonable notion of computation can be simulated by a Turing Machine.

5. Deterministic & Nondeterministic Turing Machine

6. Complexity Classes:

   - **P:** A fundamental complexity class P (or $PTIME$) is a class of decision problem that can be solved by a **deterministic** Turing machine in polynomial time.

- **EXPTIME:** A fundamental complexity class $EXPTIME$ is a class of decision problem that can be solved by a **deterministic** Turing machine in $O(2^{p(n)})$ time, where $p(n)$ is polynomial.
- **NP:** A fundamental complexity class NP is a class of decision problem that can be solved by a **nondeterministic** Turing machine in polynomial time.
- **NP(Equivalenty):** The NP decision problem has a certificate that can be **checked** by a polynomial time **deterministic** Turing machine.

7. P versus NP

8. Undecidable Problem:
   Udecidable means that there is no computer program that always gives the correct answer: it may give the wrong answer or run forever without giving any answer.
   The halting problem is the problem of deciding whether a given Turing machine halts when presented with a given input.

9. Turing's Theorem:

   The Halting problem is not decidable.

8.2 Polynomial Reduction

1. Polynomial Reduction denoted by:

$$Y \leq_p X$$

   注意：$\leq_p$ 是一个特殊符号，与 $\leq$ 没有关系，且不存在 $\geq_p$

2. To reduce problem Y to problem X (we write $Y \leq_p X$) we want a function $f : Y \to X$ that maps Y to X wuch that:

   (a) f is a polynomial time computable

(b) $y \in I_y(instance of Y) is solvable if and only if f(y) \in I_x is solvable$
注意此处对于 Y 考虑的时语言的全集，对于 X 仅需要考虑映射空间，是一个子集

3. Examples:
   Bipartite Matching $Y \leq_p X$ Max-Flow
   Circulation $Y \leq_p X$ Max-Flow

4. 如何利用 $Y \leq_p X$ 来证明一个问题的复杂度：
   对于 $Y \leq_p X$ 有：
   If we can solve X, we can solve Y
   Equivalenty, If we can **not** solve Y, then we can **not** solve X
   进一步有：If we know Y is hard, we prove that X is harder.
   注意：we reduce **TO** the problem we want to show is the harder problem.

5. NP-Hard & NP-Complete:
   X is NP-Hard, if $\forall Y \in NP$ and $Y \leq_p X$
   X is NP-Complete, if X is NP-Hard and $X \in NP$

6. Proof X is NP-Complete:
   (a) Show that X is NP
   (b) Pick a problem Y, known to be an NP-Complete
   (c) Prove $Y \leq_p X$ (reduce Y to X)

7. Boolean Satisfiability Problem: 给定一个公式，寻找一个成真的赋值
   CNF(conjunctive normal form):
   A formula is a CNF if it is a conjunction($\wedge$) of clauses.
   A clauses is a disjunction($\vee$) of literals.
   A literal is a variable or its negation.
   We can also define DNF(disconjunctive normal form) which is more easy.

8. Cook-Levin Theorem:

   Theorem, CNF SAT is NP-complete.

9. Optimization Version & Decision Version:
   Optimization: Find Max, 由于难以验证是否是最大值，最优化问题通常不是 NP 问题
   Decision: Find a solution more than k, 可以在多项式时间内验证，是 NP 问题的通常格式
   注意到对于一个多项式时间的问题，通过二分答案的方法可以将 Optimization$\leq_p$Decision，但它们仍然属于完全不同的 complexity classes

10. Proof: **Independent Set** is NP-Complete:
    首先需要证明 Independent Set 的 Decision Version 是一个 NP 问题
    其次需要找一个 NP 问题 Y 使得 $Y \leq_p X$，这里我们找 $3-SAT$，each clauses has at most 3 literal
    将 $3-SAT$ 转换为一个 Independent Set 问题，如何合理的构造图 G
    Claims: the 3-SAT instance with k clauses is satisfiable if and only if the graph G has an independent set of size k

11. Proof: **Vertex Cover** is NP-Complete:
    Theorem: for a graph G=(V,E), S is a independent set if and only if V-S is a vertex cover.
    所以，Independ Set$\leq_p$ Vertex Cover
    Claim: a graph G=(V,E) has an independent set of size at least k if and only if G has a vertex cover of size at most V-k.

12. Exercise: Proof: **Vertex Cover-Even** is NP-Complete:
    We need to proof VC$\leq_p$ VC-Even
    需要增加一个三角形，其中一个顶点连接所有其他度为奇数的顶点即可

13. Proof: **Graph Coloring**(k=3) is NP-Complete:
    需要 3 种 gadgets:Truth Gadget, Gadget for each variable, Gadget for each clause
    将 3-SAT 问题转换为 3-colorable 问题使用这三种 gadgets
    Claim: 3-SAT instance is satisfiable if and only if G is 3-colorable

14. 一个错误的证明例子：Soduku
    很容易证明 Soduku $\leq_p$ 9-colorable

且 9-colorable is NP-complete
但不能证明 Soduku 是 NP-complete，因为 $\leq_p$ 不具有可逆性，如果要证明需要**将一个 NP-Complete 问题转化为 Soduku 才可以**

15. Hamiltonian Cycle Problem(哈密顿回路)——NP-Complete
16. Traveling Salesman Problem——NP-Hard
17. Problem from Discussion:
    - Proof Hamiltonian Cycle $\leq_p$ Hamiltonian Path
      In $G = (V, E)$, random pick a vertex A
      Create vertex A_1 and A_2
      In G', $V = V - A + A_1 + A_2 + S + T$
      For each edges connect to A, create 2 edges connect to A_1 and A_2 in G'
      Connect (S, A_1) and (A_2, T)
      If a Hamiltonian Path$P$ in G', $P = S \to A_1 \to ... \to A_2 \to T$
      Then the Hamiltonian Cycle in G is $A \to ... \to A$
    - Question 3 石子移动问题，需要注意的是对于任意的图 G，拥有哈密顿路径总是比哈密顿回路普遍，所以最好将问题 reduce 到哈密顿路径而不是哈密顿回路

9 Linear Programming

9.1 Base Knowledge

1. Fundamental Theorem:
   If a linear programming problem has a solution, then it must **occur at a vertex, or corner point,** of the feasible set S associated with the problem.
   If the objective function P is optimized at **two adjacent vertices** of S, then it is optimized at **every point** on the line segment joining these vertices, in which case there are infinitely many solutions to the problem.

2. Existence of Solution:
   Suppose we are gave a LP problem with a feasible set S and a n objective function P.
   - If S is bounded, then LP has an optimal solution.
   - If S is unbounded, then may or may not be the LP problem.
   - If S is empty set, then LP has no feasible solution: infeasible.
3. The Method of Corners:
   (a) Graph the feasible set
   (b) Find the coordinates of all corner points
   (c) Find the vertex that renders the object function a maximum
4. The Standard LP form:
   $\max(c_1x_1 + ... + c_nx_n)$
   subject to:
   $a_{11}x_1 + ... + a_{1n}x_n \leq b_1$
   ...
   $a_{m1}x_1 + ... + a_{mn}x_n \leq b_m$
   $x_1 \geq 0, ..., x_n \geq 0$
5. Matrix Form:
   $max(C^TX)$
   subject to:
   $AX \leq B$
   $X \geq 0$
6. Problem $\leq_p$ Linear Programming Examples
   - Max-flow $\leq_p$ LP
   - Shortest Path $\leq_p$ LP
     Note: 使用 LP 解决最短路问题是可以**适用于负权边**的，但**只能求某一个特定点**以及其最短路径上的点的最短路，对于**其他点不保证最短路**

7. 0-1 Knapsack Problem $\leq_p$ **Integer Linear Programming(ILP)**

   Let us add an indicator variable x_k for each item:

$$x_k = \begin{cases} 1 & , Item_k Selected \\ 0 & , Item_k NotSelected \end{cases}$$

   object function:

$$max(\sum_{k=1}^{n} v_k x_k)$$

   subject to:

$$\sum_{k=1}^{n} w_k x_k \leq W$$

$$x_k \in \{0, 1\}, k = 1, ..., n$$

8. Proof ILP is NP-Hard

   Proof by: IndependentSet $\leq_p$ ILP

   Let us add an indicator variable x_k for each vertex:

$$x_k = \begin{cases} 1 & , Vertex_k BelongToIndSet \\ 0 & , Vertex_k NotBelongToIndSet \end{cases}$$

   object function:

$$max(\sum_{k=1}^{n} x_k)$$

   subject to:

$$x_i + x_j \leq 1, \forall (x_i, x_j) \in E$$

$$x_k \in \{0, 1\}, k = 1, ..., n$$

9.2 Duality

1. Definition:

   For standard maximum problem:

$$max(c^T x)$$

   subject to:

$$Ax \leq b$$

$$x \geq 0$$

The standard minimum problem is:

$$min(b^T y)$$

suject to:

$$A^T y \geq c$$

$$y \geq 0$$

2. Why convert Primal problem to Dual Problem?
   Consider the max LP constrains:

$$a_{11}x_1 + ... + a_{1n}x_n \leq b_1$$

$$\dots$$

$$a_{m1}x_1 + ... + a_{mn}x_n \leq b_m$$

Multiply by $y_k \geq 0$ for k-th equation:

$$y_1(a_{11}x_1 + ... + a_{1n}x_n) \leq b_1 y_1$$

$$\dots$$

$$y_m(a_{m1}x_1 + ... + a_{mn}x_n) \leq b_m y_m$$

Add them together:

$$y_1(a_{11}x_1 + ... + a_{1n}x_n) + ... + y_m(a_{m1}x_1 + ... + a_{mn}x_n) \leq b_1 y_1 + ... + b_m y_m$$

Collect terms wrt to x_k:

$$x_1(a_{11}y_1 + ... + a_{m1}y_m) + ... + x_n(a_{1n}y_1 + ... + a_{mn}y_m) \leq b_1 y_1 + ... + b_m y_m$$

We choose y_k in a way that:

$$A^T y \geq c$$

It follows that if:

$$(a_{11}y_1 + ... + a_{m1}y_m \geq c_1$$

then:

$$x_1c_1 \leq x_1(a_{11}y_1 + ... + a_{m1}y_m$$

so:

$$x_1c_1+...+x_nc_n \leq x_1(a_{11}y_1+...+a_{m1}y_m)+...+x_n(a_{1n}y_1+...+a_{mn}y_m) \leq b_1y_1+...+b_my_m$$

which is:

$$x_1c_1 + ... + x_nc_n \leq b_1y_1 + ... + b_my_m$$

Note that: **Max of Primal Problem $\leq$ Min of Dual Problem**

3. The **Weak Duality Theorem**:
   Let P and D be primal and dual LP correspondingly.
   If x is a feasible solution for P and y is a feasible solution for D.
   Then $c^Tx \leq b^Ty$
   Proof:

$$\begin{aligned} c^Tx &= x^Tc \\ &\leq x^T(A^Ty) \\ &= (Ax)^Ty \\ &\leq b^Ty \end{aligned}$$

4. Corollaries of Weak Duality Theorem:
   - If a Standard problem and its dual are both **feasible**, then both are **feasible bounded(F.B.)**
   - If one problem has an **unbounded** solution then the dual of that problem is **infeasible**

5. The **Strong Duality Theorem**:
   Let P and D be primal and dual LP correspondingly.
   If P and D are both **feasible**
   Then $opt(c^Tx) = opt(b^Ty)$

6. How to change equality form to inequality:
   $Ax = b$ can change to:
   $Ax \leq b$ and $-Ax \leq -b$

7. Dual problem for equality form:
   Primal Problem:
   $$max(c^T x)$$
   subject to:
   $$Ax \leq b$$
   $$-Ax \leq -b$$
   $$x \geq 0$$
   Dual problem:
   $$max(b^T y^+ - b^T y^-)$$
   subject to:
   $$A^T y^+ - A^T y^- \geq c$$
   $$y^+ \geq 0, y^- \geq 0$$
   Assume that $y = y^+ - y^-$ then:
   object function:
   $$max(b^T(y^+ y^-)) = max(b^T y)$$
   subject to"
   $$A^T(y^+ - y^-) = A^T y \geq c$$
   $$y = y^+ - y^-, unrestricted$$

8. Dual problem for general case:
   Primal Problem:
   $$max(c^T x)$$
   subject to:
   $$A_1 x \leq b_1$$
   $$A_2 x = b_2$$
   $$A_3 x \geq b_3$$
   $$x \geq 0$$
   Dual Problem:
   $$min(b_1^T y_1 + b_2^T y_2 + b_3^T y_3)$$

subject to:

$$A_1^T y_1 + A_2^T y_2 + A_3^T y_3 \geq c$$

$$y_1 \geq 0, y_2, unrestricted, y_3 \leq 0$$

9. Summary:

Primal(max)	Dual(min)
constraint$\leq$	variable$\geq$
constraint$\geq$	variable$\leq$
constraint=	variable,*unrestricted*
variable$\geq$	constraint$\geq$
variable$\leq$	constraint$\leq$
variable=	constraint,*unrestricted*

9.3 Nonlinear Optimization

1. Definition:
   object function:

$$min(f(x))$$

   subject to:

$$g_i(x) = b_i, i = 1, 2, ..., n$$

   The problem is solved using Lagrange multipliers
   核心想法：对于无约束的最优化问题，求导，找导数为 0 的点（极值点），并从中找最大值最小值。对于有约束的问题，将约束条件与目标函数以某种方式结合，并求导。

2. Lagrange Multipliers(2D case):

$$minf(x, y)$$

   subject to:

$$g(x, y) = 0$$

   Create a new function:

$$L(x, y, \lambda) = f(x, y) + \lambda g(x, y)$$

Now the problem is reduced to:

$$min_{x,y,\lambda}L(x,y,\lambda)$$

Note, that $\nabla_\lambda L(x,y)=0$, implies $g(x,y)=0$

3. Lagrange Duality:
   Primal Problem:

   $$minf(x)$$

   subject to:

   $$h_k(x)\leq 0$$

   So:

   $$L(x,\lambda)=f(x)+\sum_k \lambda_k h_k(x)$$

   Dual Problem:

   $$maxg(\lambda)=min_x L(x,\lambda)$$

   subject to:

   $$\lambda\geq 0,\forall\lambda_k$$

4. Weak Duality:
   Let P and D be the optimum of primal and dual problems respectively.
   Then $D\leq P$
   Euality(strong duality) holds for **convex functions**.

10 Approximation Algorithms

1. Why Approximation?
   Develop polynomial-time algorithms that always produce a "good enough" solution for NP-Hard problem.
   An algorithm that returns near-optimal solutions is called an **approximation algorithm**.
   We always use Greedy Approach to create Approximation Algorithm

2. Graph Coloring by Greedy Approximation

   (a) Order all vertices by degree in descending order

(b) Color the first vertex with color 1

(c) Go down the vertex list and color every vertex not adjacent to it with color 1

(d) Remove all colored vertices from the list

(e) Return to Step (b) with color 2... until all vertices is color or no more color

3. Formal Definition(For minimization problem):
   Let P be a **minimization** problem and I be an instant of P.
   Let $ALG(I)$ be the solution return by an algorithm. Let $OPT(I)$ be the optimal solution.
   Then $ALG(I)$ is said to be a $\alpha - approximation$ algorithm for:
   $\exists \alpha > 1$, if $\forall I$, $ALG(I) \leq \alpha * OPT(I)$

4. For maximization problem: $0 < \alpha < 1$ and $ALG(I) \geq \alpha * OPT(I)$

5. Vertex Cover by Greedy Approximation:
   一个用构造其他算法确认 α 的例子
   Our Approach:

   (a) Find an arbitrary edge(u, v) of E

   (b) add $\{u, v\}$ to Set C

   (c) delete edges connect to u and v, and return to (a) until E is empty

   Another Approach($2 - approximation$ algorithm):

   (a) $M \leftarrow$ Find the maximal matching on G

   (b) $S \leftarrow$ Take both endpoints of edges in M

   Theorem:
   Let $OPT(G)$ be the size of optimal vertex cover and $S = Approx(G)$
   Then, $|S| \leq 2 * OPT(G)$
   Proof: $|S| = 2|M|$ and $|M| < OPT(G)$
   Theorem:
   $\alpha = 2$ is also the tight bound of our algorithm.
   Proof: By example of bipartite graph $K_{n,n}$

6. **Metric** TSP problem:

   Definition:

   Given an undirected **Complete** graph $G = (V, E)$ with edge cost $c : e \to R^+$, find a min cost Hamiltonian cycle.

   In metric TSP, edge costs have to satisfy the **triangle inequality**. i.e. for any three vertices x,y,z, $c(x, z) \leq c(x, y) + c(y, z)$

   Metric TSP is NP-Hard by Hamitonian-Cycle $\leq_p$ Metric TSP:

$$c(x, y) = \begin{cases} 0, x = y \\ 1, (x, y) \in E \\ 2, otherwise \end{cases}$$

   Claims: The graph G has Hamitonian-Cycle iff. the TSP=$|V|$

   Greedy approximation:

   (a) Find a MST of G

   (b) Create a cycle by doubling edges

   (c) Remove double visited edges and add a **shorter** cut(by triangle inequality)

   Theorem: Approx-TSP is a $2 - approximation$ algorithm for a metric TSP

   Proof:

   |Approx-TSP| ≤ 2|MST|, by triangle inequality

   |MST| $\leq$ |OPT|, since MST at least 1 edge less then OPT

   A better approach: Christofides Algorithm:

   Note: Any graph with even degrees vertics has an **Eulerian Cycle**

   (a) $T \leftarrow$ MST of G

   (b) $S \leftarrow$ vertices of odd degree in T

   (c) $M \leftarrow$ min-cost matching on S

   (d) TSP=Euler Tour on $T \cup M$

   Without proof: Christofides is $1.5 - approximation$ of Metric TSP

7. Some NP-Hard problem **can not** find approximation algorithm.
   Example: General TSP Problem:
   Theorem:
   If $P \neq NP$, then for $\forall \alpha \geq 1$ there is NO a polynomial-time $\alpha - approximation$ for general TSP problem.
   Proof(by contradiction):
   Suppose for contradiction that such an $\alpha - approximation$ algorithm exists. We will use this algorithm to solve the Hamiltonian Cycle problem.
   In graph G:
   $$c(x,y) = \begin{cases} 1, (x,y) \in E \\ \alpha * |V|, otherwise \end{cases}$$
   If G has Hamiltonian Cycle then |TSP(G)|=|V|
   If G doesn't have HC, |TSP(G)| $\geq$ (V-1) $+\alpha * |V| > \alpha * |V|$
   Since the |TSP| differs by a factor α, our approximation algorithm can be able to **distinguish** between two cases thus decides if G has a hamiltonian cycle.
   Contradiction.
   本例主要用于证明不是所有 NP-Hard 的问题都有近似算法

8. Set Covering Problem:
   Definition: Given a collection of subsets $U = \{S_1, ..., S_n\}$ Find a min-size subset $C = \{S_i, ..., S_j\}$ such that C cover U
   Greedy Approximation Algorithm:

   (a) Find the subset S_k covers the most elements
   (b) delete all elements in S_k
   (c) return to (a) until all elements have been deleted

   Theorem:
   If the optimal solution uses k subsets, the greedy algorithm finds a solution with at most $k \ln n$ subsets
   Proof:
   Since, the optimal solution uses k subsets, there must some set that

covers at least a $\frac{1}{k}$ frachtion of it

Let us pick that set. Therefore, after the first iteration **no more than** $n - \frac{n}{k} = n * (1 - \frac{1}{k})$ left.

Again the second iteration there are not more than $n * (1 - \frac{1}{k})^2$ left

After $y = k \ln n$ iteration, there are not more than $n * (1 - \frac{1}{k})^{k \ln n} = n * (\frac{1}{e})^{\ln n} = 1$

本例主要用于描述 α 未必是常数也可以是关于 n 的函数

9. Load Balancing Problem:

   Definition: There are m machines, n jobs, for each job j has processing time t_j, Job j must run contiguously on one machine. A machine can process at most one job at a time.

   Let $J(i)$ be the subset of jobs assigned to machine i.

   The load of machine i is $L_i = \sum_{j \in J(i)} t_j$.

   The makespan is the maximum load on any machine, $L = maxL_i$

   Load balancing: assign each job to a machine to minimize makespan.

   Greedy approximation approach:

   (a) Assign job j to machine i whose load L_i is smallest.

   Theorem: Greedy algorithm is a $2 - approximation$

   Proof:

   Lemma 1: The optimal makespan $L^* \geq maxt_j$

   Lemma 2: The optimal makespan $L^* \geq \frac{1}{m} \sum_j t_j$

$$L_i - t_j \leq L_k, \forall k$$

$$\sum_{k=1}^{m} (L_i - t_j) \leq \sum_{k=1}^{m} L_k = \sum_{j=1}^{n} t_j$$

$$L_i - t_j \leq \frac{1}{m} \sum_{j=1}^{n} t_j$$

$$L_i \leq t_j + \frac{1}{m} \sum_{j=1}^{n} t_j \leq 2L^*, \forall i$$

   The approximation ratio is tight? **Not know!**

   本例主要用于说明对于一些近似算法，α 的确界不可知

11 Randomized Algorithm

1. Introduction of Probability

   - Deterministric & Randomized Algorithm
     In rondomized algorithm: $T(n) = max_{|X|=n} E[T(x)]$
   - Sample Space
   - Event: A subset of a sample space
   - Random Variable
   - Indicator Random Variable

     $$X : \Omega \to \Re, X(k) = \begin{cases} 1, k \in E \\ 0, k \notin E \end{cases}$$

   - Expectation
   - Expectation of an Indicator

     $$E[X] = Pr[E]$$

   - Linearity of Expectation

2. **Linearity of Expectation + Indicators = Your best friends forever**

3. QuickSort with Random Pivot

   - We can think of sorting as a **binary search tree rooted at the pivot**.
   - Runtime Analysis
     Let X denote the random variable counting the total number of comparisons
     For each $i < j$, we define a random variable $X_{i,j}$ that

     $$X_{i,j} = \begin{cases} 1, & i\&j\ compared \\ 0, & otherwise \end{cases}$$

Note:i,j compared only when one of them is chosen as a pivot(root)
And total number of comparisons

$$T(n) = \sum \sum_{1 \le i < j \le n} X_{i,j}$$

$$E[T(n)] = \sum \sum_{1 \le i < j \le n} E[X_{i,j}] = \sum \sum_{1 \le i < j \le n} Pr(i\, or\, j\, is\, pivot)$$

$$Pr(i\, or\, j\, is\, pivot) = \frac{2}{j - i + 1}$$

Note: 在 i→j 之间任意一个变量都有可能成为 pivot

$$E[T(n)] = \sum \sum_{1 \le i < j \le n} \frac{2}{j - i + 1} = 2(n+1)H_n - 4n$$

where $H_n = \sum \frac{1}{k} = O(\log n)$ is the n-th harmonic numbers

$$E[T(n)] = O(n \log n)$$

4. Global Min-Cut Problem

- The deterministic Approach is $O(VEV)$ in discussion
  问题定义：给定无向图 $G = (V, E)$，求对于任意 s, t，最小割的边数
  算法 1：枚举 s, t，用 FF 算法找最小割，复杂度 $O(V^2EV)$
  算法 2：注意到对于任何一个最小割，s 在其所在集合中都是等价的，所以可以固定 s 为一个随机的顶点，对 t 进行遍历即可，复杂度 $O(VEV)$
- The Randomize Approach:
  Edge Contraction: Remove an edge $e = (u, v)$, replace u, v with a single vertex, delete the self-loops and keep every other edge to the new vertex
  Complexity: $O(V^2)$
- Correctness of Randomize Approach:
  How possible if we **never** remove an edge from **real min-cut** during the algorithm?
  Let C be a min-cut of size $|C| = k$

Note, a **min degree** of any vertex in G is at least k
The number of edge $|E| = \frac{|V|*k}{2}$
Picking the first edge in min-cut: $Pr = \frac{k}{|V|*k/2} = \frac{2}{V}$ Not-Pick rate: $Pr = \frac{V-2}{V}$ For second edge: $PR = \frac{V-3}{V-2}$ Since, these event is independent:

$$Pr = \frac{V-2}{V} * \frac{V-3}{V-1} * \frac{V-4}{V-2} * ... * \frac{1}{3} = \frac{2}{V*(V-1)} = O(\frac{1}{V^2})$$

- Guessing over and over:
  After we run N times, the probability of failing all N times is

$$Pr = (1 - \frac{1}{V^2})^N$$

  Assume we run $N = 100V^2$ times $Pr = e^{-100}$ which is close to 0.
  Note, $\lim_{x->\infty}(1 - \frac{1}{x})^{ax} = e^{-ax}$
- Total Runtime Complexity of Randomize Algorithm: $O(V^4)$

5. Classification of randomized algorithms:

   (a) Las Vegas Algorithm:
   All get the correct answer, but may run for longer than expect

   (b) Monte Carlo Algorithm:
   The runtime is independent of input randomness, succeed with high probability

6. Skip Lists

   - Main Idea: **sorted linked list with shortcuts**
   - In deterministic approach:
     The number of nodes in **Level k**: $L_k = \sqrt[k]{L_1}$
     Total Complexity for searching: $k * \sqrt[k]{L_1}$
     When $k = \log_2 n$
     Total Complexity: $\log_2 n * \sqrt[\log_2 n]{L_1} = 2 * \log_2 n = O(\log n)$
     Drawback: 难以维护和实现

- Randomize approach:
  When insert an element in Level 1, flip a fair coin(50%). If it's a head - add that element to next level up and flip again.
  Search Complexity: $O(\log n)$
  Number of Level: $max\, L(X) = O(\log n)$

7. Treap(tree heap)

- Main Idea: **randomized binary search tree**
  如果输入数据是随机的，BST 通常比较平衡
  由于输入数据不是随机的，构造一个随机数，并 Rebalance BST
- Def. **A treap is a binary search tree with the heap ordering property**
- Insert:
  (a) For inserting item we get its priority at random.
  (b) Then we insert with respect to its key(BST).
  (c) Then use rotations to fix heap property with respect to its priority.
- Runtime Analysis:
  The insertion and searching is proportional to the depth.
  Let $d(k)$ denote the depth of node k.
  We need to analysis $E[d(k)]$
  We define X_{ik}:

$$X_{ik} = \begin{cases} 1, & if\, i\, is\, ancestor\, of\, k \\ 0, & otherwise \end{cases}$$

$$d(k) = \sum_{i=1}^{n} X_{ik}$$

$$E[d(k)] = \sum_{i=1}^{n} Pr[X_{ik} = 1] = \sum \frac{1}{|k-i|+1}$$

  Note, $X_{ik} = 1$ iff. node i has the highest priority from i to k

$$E[d(k)] = 2H_n = O(logn)$$

12 Final Exam Topic

1. Dynamic programming
2. Network flow and circulation
3. NP-completeness
4. Linear programming
5. Approximation
6. Randomization