Lecture 13-2
An Overall Review

1. Some homework
2. The total Content

Just for fun

- 纸上得来终觉浅，恳请老师画重点；
- 天若有情天亦老，范围一定要画小；
- 天涯何处无芳草，别说整本书都考；
- 绝知此事要躬行，老师请念师生情；
- 春眠不觉晓，挂科得补考；
- 安能摧眉折腰事权贵，题出难了我不会；
- 南朝四百八十寺，大题少让写点字；
- 桃花潭水深千尺，卷子最好一张纸；
- 谁知盘中餐，代表整个班~

Preface

- Time Complexity(时间复杂性)
- Elementary Operation
- Three Notations:
  - O(.) : "Big-Oh" – the most used
  - Ω(.) : "Big omega"
  - Θ(.) : "Big theta"

Asymptotic Notation (O)

- O-notation: asymptotic upper bound
  - $O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$.

Asymptotic Notation (Ω)

- Ω-notation: asymptotic lower bound
  - $\Omega(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$) for all $n \geq n_0\}$.

Asymptotic Notation (Θ)

- Θ-notation:
  - $\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$.

Representations of graphs

- Representations of graphs
  - Adjacency-list
  - Adjacency-matrix

Breadth first search

- Breadth first search
  - Properties of the Queue
  - Correctness
  - 22.2-7
  - Time complexity
  - Breadth first tree

Depth first search

- Depth first search
  - Properties
    - Parenthesis theorem
    - Nesting of descendants' intervals
    - White-path theorem
    - Classification of edges
  - 22.3-12
  - depth first tree

Topological sort

- Procedure
- Lemma 22.11:Acyclic $\Leftrightarrow$ no back edge
- Proof of its Correctness
- 22.4-2
- 22.4-5

Strongly Connected Component

- Procedure
- Lemma 22.13: G^{scc} is acyclic.
- Lemma 22.14: $(u, v) \in E$, where $u \in C$ and $v \in C'$. Then $f(C) > f(C')$.
- Corollary 22.15
- Theorem 22.16: Correctness
- 22.5-7
- Problem 22-3

Minimum Spanning Tree

- Tree and Its Properties
- Idea of the Generic Algorithm
- Three Properties of Minimum Spanning Tree
  - Use them to prove the correctness of some algorithm
- Theorem 23.1 : light edge is safe.
  - Use this theorem to prove the correctness of some algorithm
- Corollary 23.2: idea for Prim and Kruskal

23.1-5

- 证：反证。假设G的所有最小生成树都包含边e。任取这样的一棵最小生成树T, 在T上去掉e，将T分为两棵子树T_1和T_2。记T_1上顶点的集合为V_1，T_2上顶点的集合为V_2，则(V_1, V_2)是一个割。
- e所在的圈C至少穿越割(V_1, V_2)两次。即，C至少有两条边在(V_1, V_2)中，其中一条边是e。记e'为除e之外的另外任一条边。由题目给定，可知$w(e') \le w(e)$。
- 将e'并到T_1和T_2上，将T_1、T_2连接成一棵新的生成树T'。由于T'是在T上去掉e、加入e'形成的，可知$w(T') \le w(T)$。
- 因此，T'也是G的一棵最小生成树，且T'中不包含边e，与假设矛盾。命题得证。

Kruskal' algorithm

- Procedure
- Note:
  - sort the edges into non-increasing order
  - do if T-{e} is a connected graph
  - then T←T-e

Proof 23-4(a)

- 证明：用归纳法。假设算法过程中当前的图分别是$G_1,G_2,\ldots,G_k$, 只要证明每个G_i中都包含G的一棵最小支撑树即可。 i ＝ 1 时显然成立。假设结论对 i － 1 成立，下面证明结论对 i 也成立。假设$G_i = G_{i-1} - e$, 边e＝（u，v）. 则容易知道e 是G_{i-1}中某个圈C上的最大边。下面只需证明G_i中包含 G_{i-1}的一棵最小支撑树即可。

23-4(a)

- 证明: 设T中的边按照权值非递减顺序依次为$e_1, e_2, \ldots, e_{n-1}$,
- 即算法依次保留边$e_1, e_2, \ldots, e_{n-1}$。设边集$A_i = \{ e_1, e_2, \ldots, e_i \}$, $1 \le i \le n-1$则只需要证明每个A_i都是某棵最小生成树的子集。
- 用归纳法证明。
- i=1时，设T'是一棵最小生成树，如果$(u, v)=e_1 \in T'$, 结论自然成立。如果$e_1 \notin T'$，则在T'中存在u到v的路径p。因为删除e_1会使图不连通，即删除e_1会使顶点集合V划分为两个子集V_1和V_2，其中$u \in V_1$，$v \in V_2$。则路径p中存在1条边(x, y)满足$x \in V_1$，$y \in V_2$，并且(x, y)已经被删除了，否则如果p中所有边都没被删除，删除e_1不会使图不连通。既然(x, y)已经被删除了，根据算法是按照权值由大到小的顺序删边的，所以$w(x, y) \ge w(u, v)$。则$T'' = T' - (x, y) + (u, v)$必然是一棵最小生成树。

Continued

- 设对边集A_i时结论成立，现在证明边集A_{i+1}也是某棵最小生成树的子集。
- 设$A_i=\{e_1, e_2,\ldots, e_i\}$是最小生成树T'的子集，如果$(u, v)=e_{i+1}\in T'$，结论自然成立。如果$e_{i+1}\notin T'$，则在T'中存在u到v的路径p。因为删除$e_{i+1}$会使图不连通，即删除$e_{i+1}$会使顶点集合V划分为两个子集$V_1$和$V_2$，其中$u\in V_1$，$v\in V_2$。则路径p中存在1条边$(x, y)$满足$x\in V_1$，$y\in V_2$，并且$(x, y)$已经被删除了，否则如果p中所有边都没被删除，删除$e_1$不会使图不连通。既然$(x, y)$已经被删除了，根据算法是按照权值由大到小的顺序删边的，所以$w(x, y)\geq w(u, v)$。则$T''=T'-(x, y)+(u, v)$必然是一棵最小生成树。现在只需要证明T''包含$A_i=\{e_1, e_2,\ldots, e_{i+1}\}$中的所有边，因为T''与T'只有1条边不同，所以只需要证明$(x,y)\notin A_i$，这显然是成立的，因为(x, y)已经被删除了，而$\{e_1, e_2,\ldots, e_i\}$是没被删除的。
- 证明完毕！

23-4(c)

- 证明：算法实际上是在图G中删除一些圈上权值最重的边，最后得到一棵MST。
- 设删除的边依次为$e_1, e_2, \ldots, e_{m-n+1}$，剩余的图依次是$G_0, G_1, \ldots, G_{m-n+1}$，其中$G=G_0$，$G_{m-n+1}=T$，$m=|E|$，$n=|V|$。
- 我们证明G_{i+1}的MST同时也是G_i的MST即可。
- 前面23.1-5已经证明了存在G_{i+1}的MST T’同时也是G_i的MST，而G_{i+1}的所有MST的大小与T’一样的，所有它们都与G_i的MST的大小一样，所以他们都是G_i的MST。
- 从而G_{m-n+1}必然是$G_{m-n}, \ldots, G_o$的MST。

Prim's Algorithm

- Procedure
- Key idea

Thinking carefully

- 23-1:Second best minimum spanning tree
  - 当所有边的权重都互不相同时，MST唯一。
- 23-3: Bottleneck spanning tree

Bottleneck Spanning Tree

- A bottleneck spanning tree T of a connected, weighted and undirected graph G is a spanning tree of G whose largest edge weight is minimum over all spanning trees of G.
  - Let T_1, T_2, …, T_m are all the spanning trees G, and the largest edge of each tree is e_{t1}, e_{t2}, …, $e_{tm.}$ If $w(e_{ti}) \leq$ w(e_{tj}) for $1 \leq j \leq m$ and $j \neq i$, then T_i is a bottleneck spanning tree.
  - The value of a BST T is the weight of the maximum-weight edge in T.
  - The bottleneck spanning tree may not be unique.

BST vs MST

- Every minimum spanning tree is a bottleneck spanning tree.
  - Property 3 implies it.
  - Another proof: Let T be a MST and T' be a BST, let the maximum-weight edge in T and T' be e and e', respectively. Suppose for the contrary that the MST T is not a BST, then we have $w(e)>w(e')$, which also indicates that the weight of e is greater than that of any edges in T'. Removing e from T disconnects T into two subtrees T_1 and T_2, there must exist an edge f in T' connecting T_1 and T_2, otherwise, T' is not connected. $T_1 \cup T_2 \cup \{f\}$ forms a new tree T'' with $w(T'') = w(T) - w(e) + w(f) < w(T)$, A contradiction to the fact that T is MST, thus, a MST is also a BST.

The BST Problem

- The Problem:
  - Input: A weighted, connected and undirected graph *G*.
  - Output: A BST *T* of *G*.
- A solution:
  - Kruskal's and Prim's Algorithm works for the BST problem.
- More efficient algorithm exist for the problem?

A Verification Problem

- A verification problem:
  - Input: A graph $G = (V, E; W)$ and an integer b
  - Output: **TRUE** if the value of the bottleneck spanning tree of G is at most b and **FALSE** otherwise

How to solve this problem in linear time?

```
CHECKBOTTLENECK(G, b)
    for each vertex u ∈ V[G]
        do color[u] ← WHITE
    u ← randomly chosen vertex in G
    DFS(u, b)                                //O(V+E)
    for each vertex u ∈ V[G]
        do if color[u] = WHITE
            then return FALSE
    return TRUE
```

```
DFS(u, b)
    color[u] ← GRAY
    for each v ∈ Adj[u]
        do if color[v] = WHITE and w(u, v) ≤ b
            then DFS(v, b)
    color[u] ← BLACK
```

Solve the BST Problem

BOTTLENECK(G)

1. sort the weights of edges of G in non-decreasing order: $e[1], e[2], \ldots, e[|E|]$
2. $start \leftarrow 1$
3. $end \leftarrow |E|$
4. **while** $start < end$
5. **do** $middle \leftarrow \lfloor (start + end)/2 \rfloor$
6. **if** CHECKBOTTLENECK(G, $e[middle]$) = FALSE
7. **then** $start \leftarrow middle$
8. **else** $end \leftarrow middle$ *//in the end there is a BST with value =e[end],*
9. $u \leftarrow$ randomly chosen a vertex of G
10. call DFS(u, $e[start]$) to build a spanning tree

Solve the Problem in Linear Time

- Time complexity:
- Sorting:O(ElgE)
- Finding the value of BST: O((V+E)lgE)=O(ElgE)
- Total:O(ElgE)

- Can the problem be solved in linear time?
- Please think it carefully, anyone who finds the solution will be given an extra bonus.
- Will it be unsolved all our lives?

Single-Source Shortest Path

- Variants of Shortest-Paths Problems
- Properties of Shortest-Paths and Relaxation
  - Triangle inequality
  - Upper-bound
  - No-path property
  - Convergence property
  - Path-relaxation property
  - Predecessor-subgraph property

Bellman-Ford Algorithm

- The Bellman-Ford Algorithm
  - Correctness proof
  - 24.1-6
- In DAG
  - correctness

24.1-6

- 首先对G进行改造。增加一个新的顶点s，以及s到G中所有顶点的边，边上的权重均为0。记如此得到的图为$G' = (V', E')$。
- 将E中的边任意定一个顺序。对E中每一条边e，进行如下测试。将e从G'上去掉。调用BF算法在当前图G'上测试是否有负费用圈。若有，则e永久删除。否则，表明e在剩下的唯一一个负权重圈中，将e重新放回到G'。这样测试完E中所有的边之后，最后留在G'中的（不包括从s出发的那些新加的边）即是一个负权重圈。

Dijkstra's algorithm

- key idea
- Correctness
- Its similarity to BFS and Prim
- 24.3-2
- 24.3-3

All-Pair Shortest Paths

- Matrix multiplication
  - Idea: $l_{ij}^{(m)}$
  - The recursive formula:
  - $l_{ij}^{(1)} = w_{ij}$

    $l_{ij}^{(m)} = \min_{1 \le k \le n} \{ l_{ik}^{(m-1)} + w_{kj} \}$
  - Time complexity

The Floyd-Warshall Algorithm

- Idea: $d_{ij}^{(m)}$
- The recursive formula:
  - $d_{ij}^{(0)} = w_{ij}$
  - $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$

  $$d_{ik}^{k} = \min\{d_{ik}^{k-1}, d_{ik}^{k-1} + d_{kk}^{k-1}\} = d_{ik}^{k-1}$$

  $$d_{kj}^{k} = \min\{d_{kj}^{k-1}, d_{kk}^{k-1} + d_{kj}^{k-1}\} = d_{kj}^{k-1}$$
- Time complexity
- Run some examples
- 25.2-5

Constructing a shortest path

- For k=0

$$\pi_{ij}^{(0)} = \begin{cases} NIL & if\ i = j \text{ or } w_{ij} = \infty \\ i & \text{if } \ i \neq j \text{ and } w_{ij} < \infty \end{cases}$$

- For k>=1

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } \ d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{if } \ d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \end{cases}$$

Transitive closure

- How to revise the Floyd-Warshall Algorithm?

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \cup (t_{ik}^{(k-1)} \cap t_{kj}^{(k-1)})$$

- Time Complexity
- 25.2-8

Johnson's Algorithm

- Procedure
- Time Complexity
- Lemma 25.1: reweighting does not change shortest paths.
- Rewrite :h(v) = δ(s,v), why?
- 25.3-4

Maximum Flow

- Definitions
- There properties of the flow:
  - Capacity constraint
  - Skew symmetry
  - Flow conservation

The Ford-Fulkerson Method

- residual networks :Lemma 26.2
- augmenting paths: Lemma 26.3, Corollary 26.4
- Cuts: Lemma 26.5, Corollary 26.6
- Theorem 26.7(Max-flow Min-cut Theorem)
- Run some examples
  - f(S,T)
  - c(S,T)
- Edmonds-Karp, Dinic, Time Complexity.
- Maximum bipartite matching.

What should I say?

- We have burdened our song with so much music that it is slowly sinking and our art has become so ornate that the makeup has corroded her face.

Thank you!