QUESTION. what does \( a = [[]] \times 5 \) mean in Python?

Python 2.5.1 (r251:54863, Feb 6 2009, 19:02:12)

```python
>>> a = [1] * 5
>>> a[0] = 2
>>> a
[2, 1, 1, 1, 1]
>>> a = [[1]] * 5
>>> a[0].append(1)
>>> a
[[1, 1], [1, 1], [1, 1], [1, 1], [1, 1]]
```
**PROBLEM.** minimum spanning tree (MST) problem

- input: a set of locations $V = \{v_1, \ldots, v_n\}$, with costs for building a (undirected) link between some pairs of locations
- goal: build the cheapest communication network s.t. every pair of locations is connected (cost of a network = sum of link costs)

**FACTS.**

- removing any edge from a cycle in a connected graph leaves the graph connected.
- every $n$-node tree has exactly $n - 1$ edges.
- the complete graph $K_n$ has $n^{n-2}$ spanning trees.
**Algorithm**. Incrementally add least cost edges that do not make a cycle.

- Sort edges by cost, \( T = [] \)
- For each edge \( e \) in edges:
  - If adding \( e \) to \( T \) does not create a cycle:
    - Append \( e \) to \( T \)
- Return \( T \)

1. Add \((a, d)\)
2. Add \((c, e)\)
3. Add \((d, f)\)
4. Add \((a, b)\)
5. Add \((b, e)\)
6. Add \((e, g)\)
Kruskal’s MST algorithm

CLAIM. first $i$ selected edges are part of some MST, $i = 0, 1, \ldots, n - 1$.

- proof by induction on $i$; $i = 0$ is trivial.
- suppose first $i$ selected edges $e_1, \ldots, e_i$ are part of some MST $T$.
- done if next selected edge $e_{i+1}$ is in $T$.

OTHERWISE. adding $e_{i+1}$ to $T$ creates a cycle $C$.

- property 1: $C$ contains an edge $f \in T$ different from $e_1, \ldots, e_{i+1}$
- property 2: $c(e_{i+1}) \geq c(f)$
- remove $f$ from $C$ to form a spanning tree $T'$
- $c(T') = c(T) + c(e_{i+1}) - c(f) \leq c(T)$
- thus $T'$ is a MST containing $e_1, \ldots, e_{i+1}$
Kruskal's MST algorithm

**Algorithm.** Incrementally add least cost edges that do not make a cycle

1. Sort edges by cost, \( T = [] \)
2. For each edge \( e \) in edges:
   - If adding \( e \) to \( T \) does not create a cycle:
     - Append \( e \) to \( T \)
3. Return \( T \)

Rewriting the for loop:

1. For each edge \( (u,v) \) in edges:
   - If \( u \) and \( v \) are in different connected components in \( T \):
     - Append \( e \) to \( T \)

Need to maintain connected components of \( T \) with efficient “merge”
**Kruskal’s MST algorithm**

**UNION-FIND.** data structure for disjoint sets with efficient merge

- **MAKEUNIONFIND**(S) initializes with all elements of S in separate sets
- **FIND**(u) returns the name of the set containing u
- **UNION**(A, B) merges two sets A and B

**IMPLEMENTATION (KRUSKAL’S).** use union-find for intermediate T

sort edges by cost

T = makeunionfind(nodes) ## start with n connected components

for e=(u,v) in edges:
  if T.find(u) != T.find(v) ## adding e does not create a cycle
    T.union(T.find(u), T.find(v)) ## merge components for u, v

return T

- **Running time:** $O(m \log m) = O(m \log n)$ plus **MAKEUNIONFIND** and $O(m)$ **FIND** and $O(n)$ **UNION** operations
**Kruskal’s MST algorithm**

**UNION-FIND.** data structure to maintain disjoint sets (e.g. connected components of a graph) and supports merge operations

- **MAKEUNIONFIND(S)** initializes with all elements of $S$ in separate sets
- **FIND(u)** returns the name of the set containing $u$
- **UNION(A, B)** merges two sets $A$ and $B$

**IMPLEMENTATION.** simple array-based union-find

- maintain an array **component** that maps $u$ to set containing $u$
- each disjoint set is named after a “representative” element
- **MAKEUNIONFIND(S)**: set component[$s$] = $s$ for all elements of $S$
- **UNION(A, B)**: name $A \cup B$ after the larger set (requires $\min(|A|, |B|)$ updates to **component**)

**RUNNING TIME.**

- **MAKEUNIONFIND(S)**: $O(n)$ time where $n = |S|$
- **FIND(u)**: $O(1)$ time
- **UNION(A, B)**: $O(k \log k)$ time for any sequence of $k$ such operations
Prim’s MST algorithm

ALGORITHM. build a single tree incrementally by adding least cost edges

\[ S, T = [s], [] \] # pick any start vertex \( s \)

while len(T) != n-1:
    let e be cheapest edge with exactly one end-point in S
    T.append(e)
    S.append(end-point of e outside S)

return T

1. add \((a, d)\)
2. add \((d, f)\)
3. add \((a, b)\)
4. add \((b, e)\)
5. add \((c, e)\)
6. add \((e, g)\)
Prim’s MST algorithm

IMPLEMENTATION. use a priority queue (ala Dijkstra’s).

▶ store nodes outside $S$ in a priority queue $H$
▶ key of a node $u$ is cost of cheapest edge connecting $u$ to some node $v$ in $S$

\[
\text{while } \text{len}(T) \neq n-1:\n\]
\[
u = H.\text{extractmin}()\]
\[
T.\text{append}((u,v))\]
\[
S.\text{append}(u)\]
\[
\text{for } w \text{ in } \text{nb}[u]:\n\]
\[
\text{if } w \text{ not in } S:\n\]
\[
H.\text{changekey}(w, \ldots)\]

\text{return } T

▶ running time: $O(m + n)$ plus $n$ EXTRACTMIN and $m$ CHANGEKEY operations
▶ can implement CHANGEKEY in min-heap with INSERT/DELETE or HEAPIFY-UP/HEAPIFY-DOWN along with a reverse look-up table
▶ total running time: $O(m \log n)$
PROBLEM. divide \( n \) objects into \( k \) groups (a \( k \)-clustering) so that objects in different groups are as far apart (different) as possible.

- e.g. categorizing documents, grouping customers based on movie preferences

INPUT. \( n \) objects \( p_1, \ldots, p_n \), parameter \( k \), and a distance measure \( d(\cdot, \cdot) \)

- \( d(p_i, p_i) = 0 \) and \( d(p_i, p_j) > 0 \) for \( i \neq j \); \( d(p_i, p_j) = d(p_j, p_i) \)

GOAL. partition \( p_1, \ldots, p_n \) into \( k \) non-empty sets with maximum spacing

- spacing: minimum distance between any point of points in different clusters (measures how far apart the clusters are)
**ALGORITHM.** Start with \( p_1, \ldots, p_n \) in \( n \) different clusters and keep merging closest clusters until we have \( k \) clusters (like Kruskal’s).

- Sort pairs of points by distance.
- \( C = \text{makeunionfind}(\text{points}) \)
- For (p,q) in pairs:
  - If \( C.\text{find}(p) \neq C.\text{find}(q) \)
    - \( C.\text{union}(C.\text{find}(p), C.\text{find}(q)) \)
  - If \( C.\text{size}() == k \):
    - Return \( C \)

- Running time: \( O(n^2 \log n) \)
- Same as computing a MST and deleting the \( k - 1 \) most expensive edges.
THE END