



CS3233 Competitive Programming

Steven Halim
Graph (customized for IOI syllabus)

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

429 – Word Transformation (1)

- Given starting, ending, and list of other words
- Find **shortest** sequence of 1 char transformation from starting word to ending word
 - Example:
 - Starting word: **spice**, Ending word: **stock**
 - List of other words (max 200): dip lip mad map maple may pad pip pod pop sap sip slice slick spice stick stock
 - Ans: spice slice slick **stick** **stock** (4 transformations)
- No graph in problem description?

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Outline

- Graph modeling & identifying keywords
- Interesting applications of graph algorithms
 - SP, DFS, MST problems
 - No max flow, no bipartite graphs... ☹
- Graph versus DP
- Special graphs
- Admin
 - Presentation & “challenge each other” contest

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

429 – Word Transformation (2)

- Where is the graph?
 - Each word is a vertex
 - Connect two vertices (words) with edge if Hamming distance between them is 1
 - (only 1 character difference)
- What is the graph problem?
 - **sssp** from starting word, output dist[ending word]
- What is the appropriate graph algorithm?
 - $O(V + E)$ BFS, as the graph is unweighted

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Graph Modeling

- The “hardest part” of solving graph problems may be the “modeling part”
 - Read the question
 - Find the graph! (maybe hidden)
 - Define the graph problem
 - May need some tweaks to make it more familiar...
 - Find appropriate graph algorithm
 - Code it
- PS: Also applicable for other problem types...

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Applications...

SHORTEST PATHS PROBLEMS

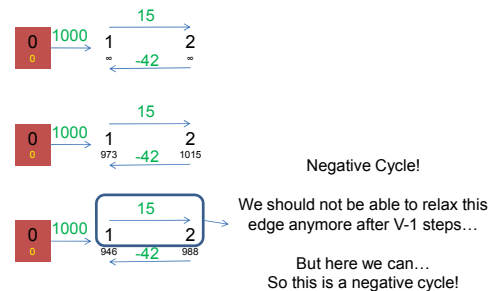
CS3233 - Competitive Programming,
Semester 2, 2009/10, SoC, NUS

558 – Wormholes

- There are X wormholes in star system.
- If you enter wormhole i that is connected to wormhole j , you can either teleport $+x$ years to the future or $-y$ years to the past.
- There are $X \leq 1000$ wormholes and 2000 'time tunnels'.
- Question: **Can you go back to the beginning of universe?**

CS3233 - Competitive Programming,
Semester 2, 2009/10, SoC, NUS

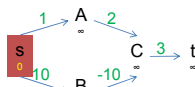
Visualization – 558



CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Single Source Shortest Paths

- If the graph has **negative edge weight**
 - Dijkstra fails
- Solution: Bellman Ford
 - Complexity: $O(V^3)$ or $O(V * E)$
- Bellman Ford can also deal with **negative cycle**
- UVa: [558](#) (Wormholes)
 - Bellman Ford can detect negative cycle (but this shortest paths problem cannot be solved)



CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

11492 – Babel (1)

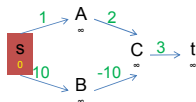
- Given start, finish language, and list of words that are common in two languages
 - e.g. the word “red” is a “color” in English, but it means “network” in Spanish
- Find chain of words that satisfy the lang links, 1st char rule, and min its num of total chars:
 - Lang links: $w_1 (L_{start}/L_a), w_2 (L_a/L_c), \dots, w_i (L_x/L_{finish})$
 - 1st char rule: $w_1[0] \neq w_2[0]$
 - Total length of chars: $len(w_1) + len(w_2) + \dots len(w_i)$

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Bellman Ford

```
memset(dist, INF, sizeof(dist)); dist[source] = 0;
REP (i, 0, n - 1) // relax edges repeatedly
  REP (u, 0, n - 1)
    TR (G[u], v) // has edge and can be relaxed
      dist[v->first] = min(dist[v->first],
                           dist[u] + v->second);

negative_cycle_exist = false;
REP (u, 0, n - 1) // one more pass to check
  TR (G[u], v)
    if (dist[v->first] > dist[u] + v->second)
      negative_cycle_exist = true;
```



CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

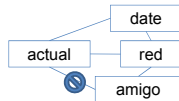
11492 – Babel (2)

- See example for clarity
 - Example (max 2000 words):
 - Start language: portugues, Finish language: frances
 - Words: ingles espanhol **red**, espanhol portugues **amigo**, frances ingles **date**, espanhol ingles **actual**
 - Infeasible 1: amigo date, length = 9
 - Infeasible 2: amigo actual date, length = 15
 - Not optimal: amigo red actual date, length = 18
 - Optimal: amigo red date, total length = 12

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

11492 – Babel (3)

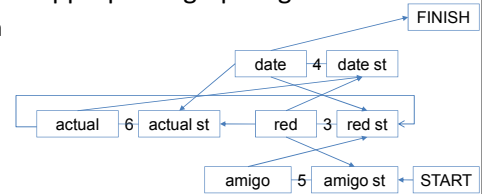
- Where is the graph?
 - Each word is a vertex
 - Connect two vertices (words) with edge if they share common language & have different 1st char!
- But...
 - There can be many possible sources and destinations?
 - Many words of starting and finish language



CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

11492 – Babel (6)

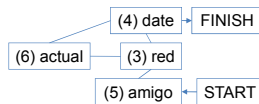
- Now what is the graph problem?
 - sssp on weighted graph with non negative edge weight from START to FINISH
- What is the appropriate graph algorithm?
 - Dijkstra



CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

11492 – Babel (4)

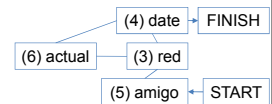
- Where is the graph? – continued
 - Introduce “dummy vertices”: START and FINISH
 - Connect “START” to all words that are inside the vocabulary of start language
 - Connect all words that are inside the vocabulary of finish language to “FINISH”
 - Seems like sssp, but...
 - There is no edge weight
 - But we have vertex weight of len(word)?



CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

11492 – Babel (7)

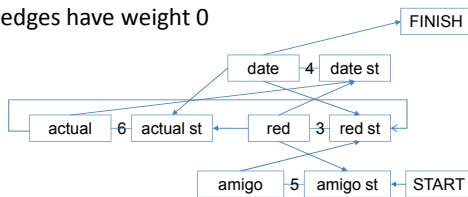
- However...
 - We can simply use modified Dijkstra on this graph
 - Although all edges (v,u) have cost = 0, we can say that:
 $D[u] = D[v] + 0 + \text{wordlen}[u]$



CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

11492 – Babel (5)

- Where is the graph? – continued
 - Introduce “split vertices”
 - A vertex with weight X can be split into 2 vertices connected with edge of weight X
 - Normal edges have weight 0



CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

11463 – Bombing Mission

- Given: a table that stores the amount of minutes to travel between buildings (there are at most 100 buildings) and two special buildings: startB and endB.
- There are K soldiers who have to bomb all the K buildings in this mission.
- Each of them start at the same time from startB, choose one building B that has not been bombed by other soldier (bombing time negligible), and then gather in (destroyed) building endB.
- What is the minimum time to complete the mission?

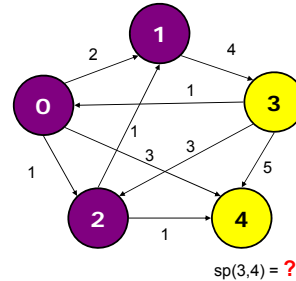
CS3233 - Competitive Programming, Semester 2, 2009/10, SoC, NUS

All Pairs Shortest Paths (1)

- UVa: [11463](#) (Commandos)
 - How long do you need to solve this problem?
- Solution:
 - The answer is determined by **sp** from starting building, detonate **furthest building**, and **sp** from that furthest building to end building
 - $\max(\text{dist}[\text{start}][i] + \text{dist}[i][\text{end}])$ for all $i \in V$
- How to compute **sp** for **many** pairs of vertices?

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Floyd Warshall – Example (1)



Suppose we want to know the shortest path between vertex 3 and 4, i.e. $\text{sp}(3, 4)$

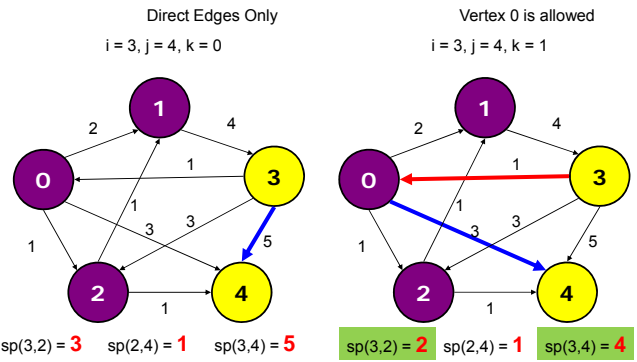
$\text{sp}(3,4) = ?$

All Pairs Shortest Paths (2)

- This problem is called: All Pairs Shortest Paths
- Two options to solve this:
 - Call SSSP algorithms multiple times
 - Dijkstra $O(V * (V+E) * \log V)$, if $E = V^2 \rightarrow O(V^3 \log V)$
 - Bellman Ford $O(V * V * E)$, if $E = V^2 \rightarrow O(V^4)$
 - Slow to code
 - Use Floyd Warshall, a clever **DP** algorithm
 - $O(V^3)$ algorithm
 - Very easy to code!

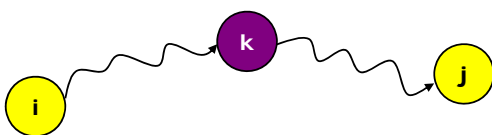
CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Floyd Warshall – Example (2)

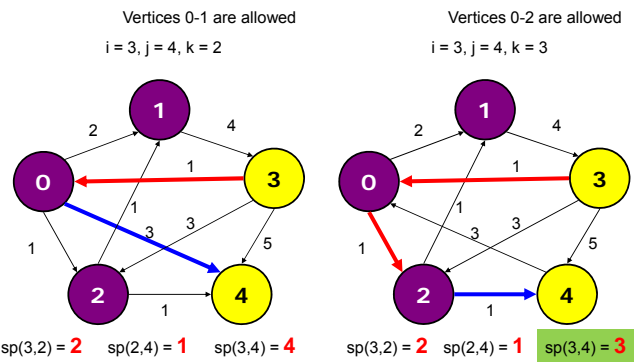


Floyd Warshall – Idea (1)

- Label the vertices as $[0 .. N-1]$
- Restrict $\text{sp}(i, j)$ to consist of vertices $[0 .. k-1]$ only (except i and j)
- Iteratively relax k from 0 to $N-1$



Floyd Warshall – Example (3)



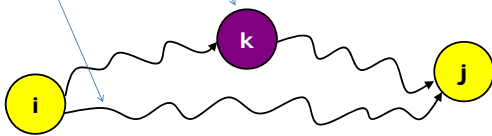
Floyd Warshall – Idea (2)

$D_{i,j}^0$: Edge weight of the original graph

$D_{i,j}^k$: Shortest distance from i to j involving $[0..k-1]$ only as intermediate vertices

$$D_{i,j}^k = \begin{cases} w_{i,j} & \text{for } k = 0 \\ \min(D_{i,j}^{k-1}, D_{i,k}^{k-1} + D_{k,j}^{k-1}) & \text{for } k > 0 \end{cases}$$

Not using vertex k Using vertex k



Floyd Warshall Variants (1)

- Floyd's algorithm is to get all pairs shortest paths – But we can have some other variants of usage^

- Two examples:

1. Transitive Closure (Warshall's algorithm): 334

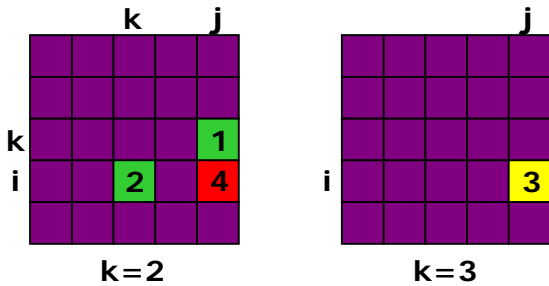
- We just want to see if vertex i is connected to j

```
REP (k, N) // O(n^3)
  REP (i, N)
    REP (j, N)
      d[i][j] = d[i][j] | (d[i][k] & d[k][j]);
```

- Logical bitwise operation is faster than arithmetic operation

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

The DP Tables



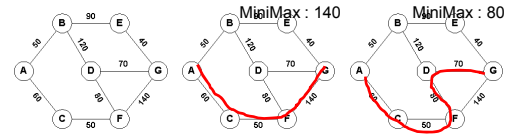
Floyd Warshall Variants (2)

2. MiniMax: 534, 10048; & its reverse MaxiMin: 544, 10099

- For a path from i to j , we pick the max edge weight
- We want to minimize max edge weight among all possible paths from i to j

```
REP (k, N) // O(n^3)
  REP (i, N)
    REP (j, N)
      d[i][j] = min(d[i][j], max(d[i][k], d[k][j]));
```

Find path from A to G that satisfy "MiniMax" criteria



CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Floyd Warshall – Template

- $O(V^3)$ since we have three nested loops!
- Use adjacency matrix – So that weight of edge(i, j) can be accessed quickly
- Can use 2-dimensional matrix (instead of 3-d) – Dimension k can be done "on the fly"

```
REP (k, 0, n - 1)
  REP (i, 0, n - 1)
    REP (j, 0, n - 1)
      G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
```

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Applications

DFS STUFFS

CS3233 - Competitive Programming, Semester 2, 2009/10, SoC, NUS

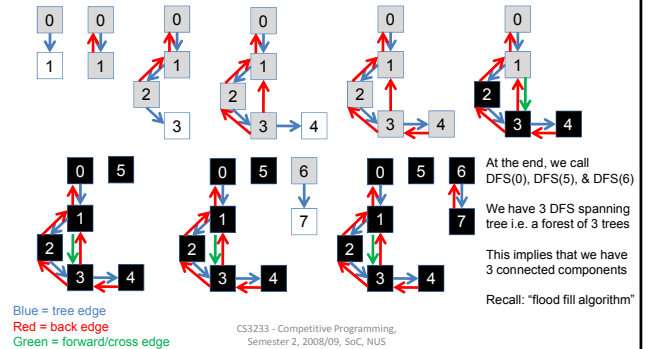
Selection Test 1 Problem C

- Given Singapore road map, sabotage either an **intersection** or a **road** that has minimum cost s.t. Singapore road network breaks down.
- This is problem of finding Articulation point & Bridges
 - Solvable using $O(V+E)$ DFS

CS3233 - Competitive Programming, Semester 2, 2009/10, SoC, NUS

DFS Animation

Assume that we start from DFS(0), neighbors are in ascending order



DFS Spanning Tree/Forest

- When DFS is executed on a graph, it will form DFS spanning tree (or forest)
- With it, we can classify edges into four types:
 - Tree edges: those selected/traversed by DFS
 - Back edges: for testing cycle
 - Forward edges: connect vertex to its descendant
 - Cross edges: all other edges

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Finding Articulation Points (1)

- Articulation Point (UVA: [315](#), [10199](#)):
 - A vertex in graph G whose removal disconnects G
 - Graph without articulation point is called "Biconnected"
- Trivial Algorithm: $O(V * (V+E)) = O(V^2 + VE)$
 - Run $O(V+E)$ DFS to count number of connected components (cc) of the original graph
 - Repeat for all vertex $O(V)$
 - Cut (remove) one vertex v and its incident edges
 - Run DFS to check if number of cc increase $O(V+E)$
 - If yes, v is an articulation point/cut vertex
 - Restore v and its incident edges

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Code: White-Gray-Black DFS

```
void dfs(int u) { // DFS with some book-keeping...
    color[u] = GRAY; // a little addition from Week 05 code!
    dfsnum[u] = counter++; // counter = global var, init = 0

    TR (G[u], v) {
        if (color[*v] == WHITE) // GRAY to WHITE
            dfs(*v); // edge (u, *v) is a TREE edge
        else if (color[*v] == GRAY) // GRAY to GRAY
            printf("edge (%d, %d) is a back edge (cycle)\n", u, *v);
        else if (color[*v] == BLACK) // GRAY to BLACK
            printf("edge (%d, %d) is a forward/cross edge\n", u, *v);
    }

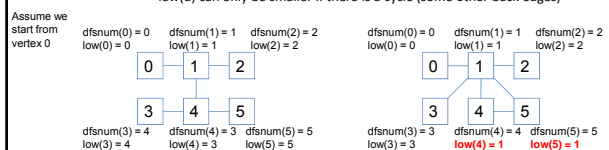
    color[u] = BLACK;
}
```

Note: you may want to learn non-recursive DFS to avoid stack overflow error

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

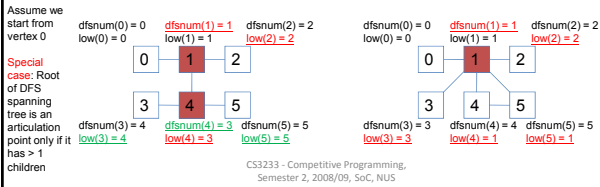
Finding Articulation Points (2)

- Better Algorithm: Just modify the $O(V+E)$ DFS
 - Run DFS, but now we count $dfsnum(u)$ and $low(u)$
 - $dfsnum(u)$ = iteration counter when u is **first** visited
 - $low(u)$ = lowest $dfsnum$ directly reachable from u
 - Initially $low(u) = dfsnum(u)$ when u is first visited
 - Do not update $low(u)$ with back edge (u, v) if v is a direct parent of u
 - $low(u)$ can only be smaller if there is a cycle (some other back edges)



Finding Articulation Points (3)

- Better Algorithm: Just modify the $O(V+E)$ DFS
 - Now, when we are in a vertex u where $dfsnum(u) \leq low(v)$ and v is a neighbor of u , then u is an articulation vertex
 - The fact that $low(v)$ is not smaller than $dfsnum(u)$ imply that there is no back edge to vertex w that has lower $dfsnum(w)$
 - To reach parent of u from v , one must pass through u
 - Removing vertex u will thus disconnect the graph



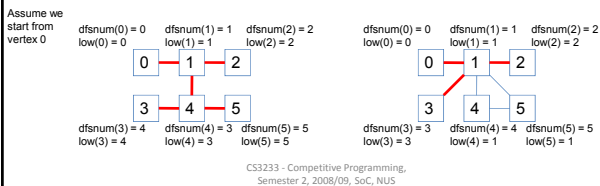
How About BFS Spanning/SP Tree?

- Nothing much...^
 - Potential application: to reconstruct all shortest paths from single source of an un-weighted graph

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

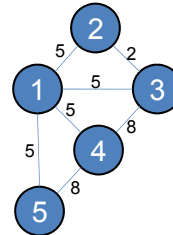
Finding Bridges

- Bridge (UVa: [796](#), [610](#)):
 - An edge in graph G whose removal disconnects G
- Simple modification from previous DFS code
 - When $dfsnum(u) < low(v)$, then edge $u-v$ is a bridge
 - Similar reasoning as previous slide



How to Solve This?

- Given this graph, select some edges s.t that makes the graph connected but with minimal total weight!



- MST!

CS3233 - Competitive Programming, Semester 2, 2009/10, SoC, NUS

Code: Modified DFS

```
void dfs(int u) { // init: all parent = dfsnum = -1, counter = 0
    low[u] = dfsnum[u] = counter++; // low[u] is at least dfsnum[u]
    TR (G[u], v)
    if (dfsnum[*v] == -1) { // a tree edge
        parent[*v] = u;
        if (u == cur_root)
            children++; // count children of root
        dfs(*v);
        if (low[*v] >= dfsnum[u]) // for bridge, set to >
            articulation_vertex[u] = true; // for bridge, edge (u,*v)
        low[u] = min(low[u], low[*v]); // update low[u] if possible
    }
    else if (*v != parent[u]) // a back edge and not direct cycle
        low[u] = min(low[u], dfsnum[*v]); // back edge, update low[u]
    } // final check: articulation_vertex[cur_root] = (children > 1);
}
```

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Tree

- Tree is a subset of Graph
 - V vertices and exactly $E = V - 1$ edges
 - No cycle, one unique path between two vertices
 - Sometimes, it has one special vertex called "root" (rooted tree)
 - Root has no parent
 - A vertex in n -ary tree has either $\{0, 1, \dots, n\}$ children
 - $n = 2$ is called binary tree (most popular one)
 - Has many applications:
 - Organization Tree, Directories in Operating System, etc

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Spanning Tree & MST

- Given a **connected undirected** graph G , Select $E \in G$ such that a tree is formed and this tree **spans** (covers) all $V \in G$!
 - No cycles or loops are formed!
- There can be **several** spanning trees in G
 - The one where total cost is minimum is called the **Minimum Spanning Tree (MST)**
- UVa: [908](#) (Re-connecting Computer Sites)

Kruskal's Algorithm

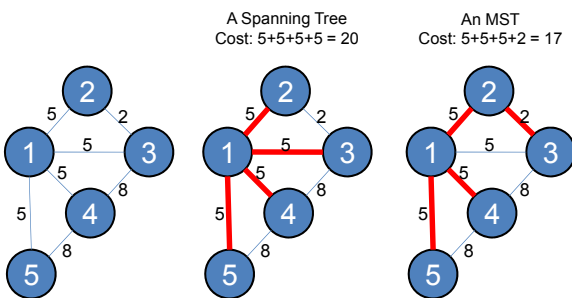
- In my opinion, Kruskal's algorithm is simpler


```
while there are unprocessed edges left
  pick an edge e with minimum cost
  if adding e to MST does not form a cycle
    add e to MST
```

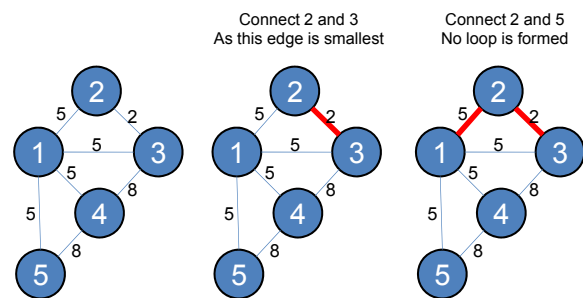
 - Either use Priority Queue or simply sort the edges
 - No need to modify priority_queue in STL <queue>
 - This is an issue for Prim's algorithm
 - Test for cycles using Disjoint Sets (Union Find) DS

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Visualization – 908 (1)



Visualization – 908 (2)

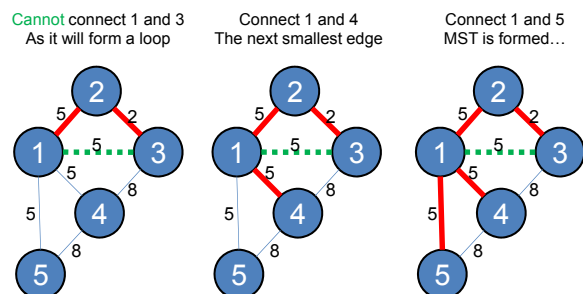


Algorithms for Finding MST

- Prim's (Greedy Algorithm)
 - At every iteration, choose an edge with minimum cost that does not form a cycle
 - "grows" an MST from a root
- Kruskal's (also Greedy Algorithm)
 - Repeatedly finds edges with minimum costs that does not form a cycle
 - forms an MST by connecting forests
- Which one is easier to code?

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Visualization – 908 (3)

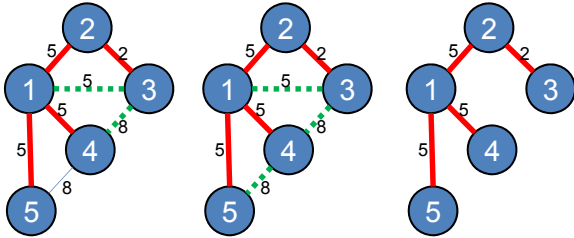


Visualization – 908 (4)

But Kruskal's algorithm will still continue

However, it will not modify anything else

This is the final MST with cost 17



MST Variants (2)

2. Minimum Spanning Forest (UVA: [10369](#))

- Spanning: All vertices must be covered
 - But we can stop even though the Tree has not been formed as long as the spanning property is satisfied!
- The desired number of components is told beforehand
 - Result is a “forest”
- Solution?
 - Use Kruskal's algorithm again

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Kruskal's Algorithm (Sample Code)

```
// sorted by edge cost, PQ default: sort descending :(
priority_queue< pair<int, ii> > EdgeList;
... // trick: store (negative) weight(i, j) and pair(i, j)
initSet(); // all V are disjoint sets at the beginning
new_cost = 0; // the cost of MST

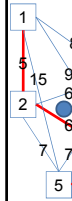
while (!EdgeList.empty()) { // while ∃ more edges
    pair<int, ii> front = EdgeList.top(); EdgeList.pop();
    if (!sameSet(front.second.first, front.second.second)) {
        // if adding e to MST does not form a cycle
        new_cost += (-front.first); // add -weight of e to MST
        unionSet(front.second.first, front.second.second);
    }
}
```

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

MST Variants (3)

3. MST for Comp. Geometry

- See CS3233 Contest 3 Problem C (Fat King):
 - Given $|V|$ pillars, and all $|V|^2$ possible edges
 - MST selects $|V|-1$ shortest edges that **do not** form a cycle
 - Adding **chord**, a non MST edge, to this MST will cause a cycle!
 - If this cycle traps origin (0, 0), where Pudge's bed is located
 - » This chord X will be (one of) the longest edge in cycle
 - » If X is the shortest so far, memorize it
 - Do this for all possible chords to obtain the answer^
- This idea of using MST and its chords *may be* used for other similar Comp. Geometry problems



CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

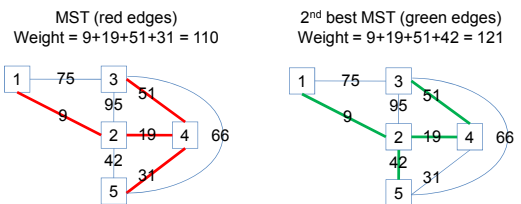
MST Variants (1)

- MST can be more complex than its basic form:
- Two MST variants:
 1. “Partial” MST (UVA: [10147](#), [10397](#))
 - Some edges are already **fixed**
 - Must be taken as part of the Spanning Tree
 - We need to continue building the “M”ST
 - The resulting Spanning Tree perhaps no longer minimum
 - Solution?
 - Use Kruskal's algorithm to continue!

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

MST Variants (4) – (1)

4. 2nd-best MST (UVA: [10600](#), PKU: [1679](#)), example:



2nd best MST is always MST with 2 edges difference.
1 edge is taken out from MST, another **chord** edge is added to MST!
In this example: edge (4-5) → taken out and (2-5) → added in.

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

MST Variants (4) – (2)

Remove edge (1-2)
MST Weight = 176

Remove edge (3-4)
Weight = 136

Remove edge (2-4)
MST Weight = 133

Remove edge (4-5)
Weight = 121

Simple solution^A: Sort edges $O(E \log E)$, find MST using Kruskal in $O(E)$, then for each edge in MST, make its weight INF ("delete"), and try to find the (2nd best) MST $O(VE)$. It is V^2E because $|E|$ in MST is $|V|-1$

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Note: similar idea can be used to find 2nd best shortest path

MST Variants (5) – (3)

Restore the edges incident to constrained vertex and try to add these edges to MST, lowest to largest weight. Whenever a cycle appears, remove edge with largest weight in that cycle...

We stop when vertex with degree constraint has reached its maximum degree or when no more edge can decrease the MST weight*

Weight = $65+19+32+43+24 = 183$
Still the same as with normal MST

Degree constraint of Park = 3

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

MST Variants (5) – (1)

5. 1st vertex has degree constraint ([L. Archive 2099](#))

Degree constraint of Park = 3

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

MST Variants (5) – (4)

If "Park" has degree constraint = 1, simple MST algorithm will fail. We must use the approach outlined previously!

If we apply normal MST algorithm, we will "accidentally" select edge "Clemenzi-Park" and "Herb-Park", which will violate the degree constraint of "Park"

Weight = $82+19+32+43+79 = 255$

Degree constraint of Park = 1

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

MST Variants (5) – (2)

Remove edges incident to the constrained vertex and find Minimum Spanning Forest on the other vertices

Degree constraint of Park = 3

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

MST Variants (6)

6. Minimize average cost/distance ([L. Archive 3465](#))

- For each edge, there is a length $L(i)$ and a cost $C(i)$
 - Minimize: $d = \frac{\sum(C(i) \text{ in MST})}{\sum(L(i) \text{ in MST})}$
 - Rewritten as: $0 = \sum(C(i) \text{ in MST}) - d * \sum(L(i) \text{ in MST})$
 $0 = \sum(C(i) - d * L(i) \text{ in MST})$
 - If we use $(C(i) - d * L(i))$ as the edge weight (maybe negative), we can get an MST with cost 0
 - We use binary search to get d
 - For a d' , we use $C(i) - d' * L(i)$ as the edge weight, and generate the MST
 - If $\text{cost}'(\text{MST}) < 0$, we reduce d'
 - If $\text{cost}'(\text{MST}) > 0$, we increase d'

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

MST Variants (7) – (1)

7. MiniMax/MaxiMin

- Recall Floyd Warshall Variants
 - This solution requires $O(V^3)$

Floyd Warshall Variants (2)

2. MiniMax: [535_10048](#); & its reverse MaxiMin: [544_10099](#)

- For a path from i to j , we pick the max edge weight
- We want to minimize max edge weight among all possible paths from i to j

```

REPRI (i, j) // O(N^2)
REPRI (i, j)
  d[i][j] = min(d[i][i], max(d[i][k], d[k][j]))
  
```

Find path from A to G that satisfy "MaxiMin" criteria

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Graph versus DP

- Some graph problems may be disguised as (or looks like) non DP
 - e.g. problem looks like a shortest path with some constraints on graph, but the constraints fail *greedy* SSSP algorithm!
 - Discuss
 - <http://www.spoj.pl/problems/FISHER/>
 - CS3233 Contest 1 Problem D (Duilian)

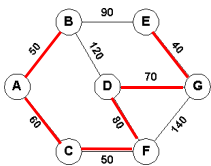
CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

MST Variants (7) – (2)

For MiniMax, we can build an MST of the graph first
MST is a connected!

- There will be a unique path from any two vertices A and B
- Answer: pick the maximum edge along the path A to B

It is the other way around for MaxiMin



The answer from A-G is the max edge along edges in MST: A-C-F-D-G, which is 80

This is the same answer as with $O(V^3)$ Floyd Warshall version

Kruskal runs in $O(E \log V + E)$, plus $O(V)$ DFS/BFS on the MST...

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Trees, Directed Acyclic Graph, Bipartite Graphs (Omitted)

SPECIAL GRAPHS

CS3233 - Competitive Programming,
Semester 2, 2009/10, SoC, NUS

GRAPH OR DP?

CS3233 - Competitive Programming,
Semester 2, 2009/10, SoC, NUS

Problem Involving Tree DS (1)

- Given a **Tree T** with **V** vertices
 - Each vertex v has weight $W(v)$
- Task:
 - Pick **subset** of vertices in **T** that are independent
 - If vertex a and b are selected, edge(a - b) is **not** in **T**!
 - Also maximize sum of weights of selected vertices!
- Maximum Weight Independent Set (MWIS)!
 - To be precise, MWIS on a Tree

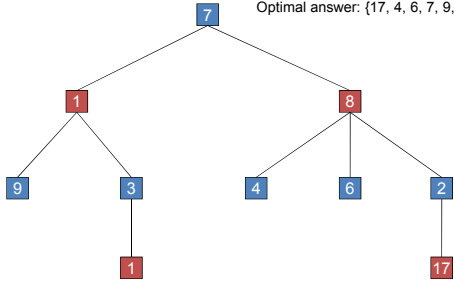


66

Problem Involving Tree DS (2)

• Example

If we select {1, 8, 1, 17}, this is an independent set, with total weight 1 + 8 + 1 + 17 = 27
Can you spot a better answer?
Optimal answer: {17, 4, 6, 7, 9, 3}. Total = 46

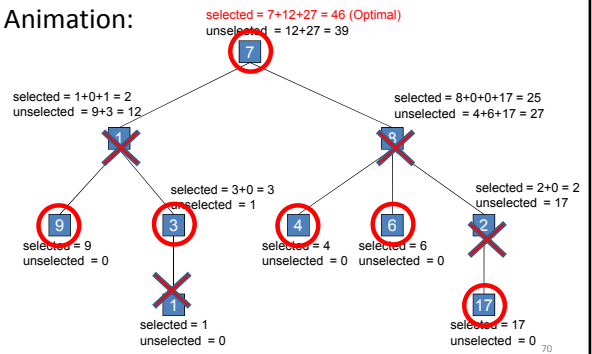


67

Solution: DP on Tree (3)

Solution can be implemented either in top down or bottom up fashion. There are only 2^V states, so DP runs in O(V)!

• Animation:



70

Solution: DP on Tree (1)

• Important Tip:

- Almost for all tree-related problems, we need to "root the tree" first
 - Potentially the subproblems w.r.t subtrees appear!

• So, how to formulate the DP?

- If we have a rooted (sub)tree, can we formulate a MWIS recurrence w.r.t their children?
- One formulation: $C(v, selected) = \max$ weight of subtree rooted at v if it is $selected = [true/false]$!

68

MWIS can be harder...

• Resource Allocation Problem

- Abridged Problem Description:
 - There are two users: User A and B
 - Each user has transactions, e.g. A has $\{A_1, A_2, \dots, A_n\}$
 - Each transaction has weight, e.g. $W(A_1), W(A_2)$, etc
 - Transactions use shared resources: e.g. A_1 uses $\{r_1, r_2\}$
 - Access to resource is exclusive, if A_1 is selected, then user B's transaction(s) that use either r_1 or r_2 cannot be selected
 - A_1 and A_2 will never use same resource
 - » But A_1 and B_1 may be
- Maximize the sum of weight of selected transactions!

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Solution: DP on Tree (2)

• Base case (the leaves: obvious)

- $C(v, false) = 0$ and $C(v, true) = w(v)$ for all leaves v

• Recurrence relation (the internal nodes)

- $C(v, true) = \sum_{c \in \text{Children}(v)} C(c, false) + w(v)$
 - If root v is taken, children of v are not taken
- $C(v, false) = \sum_{c \in \text{Children}(v)} \max(C(c, false), C(c, true))$
 - If root v is not taken, children of v maybe taken (or not)

• Answer will be in the selected root

- $\max(C(\text{root}, false), C(\text{root}, true))$

69

MWIS (Again)... But NOT on Tree!

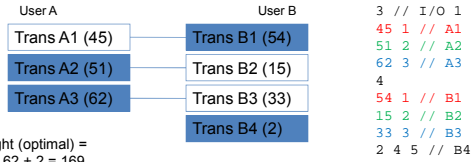
• Finding solution by identifying keywords:

- If a transaction from user A is selected, then transactions from user B that **share** some or all resources **cannot** be selected \rightarrow Independent Set
- Maximize sum of weight of selected transactions \rightarrow Maximum Weighted Independent Set (MWIS)
- There are only two users \rightarrow 2 sets
- No conflict between 1 user \rightarrow Bipartite
- So: MWIS on Bipartite Graph

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

MWIS on Bipartite Graph

- Let's look at simple test case (Sample I/O 1)



Total weight (optimal) = 54 + 51 + 62 + 2 = 169

- Can we use $O(V)$ DP as with MWIS on Tree?
 - Do not think so..., the states are too complex

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

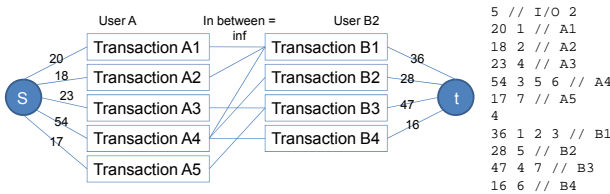
"Off Topic 1": Dinic Algorithm

- There are few ways to find augmenting paths
 - Find any augmenting path: use DFS
 - Ford Fulkerson, slow $O(E * |MaxFlow|)$
 - Shortest augmenting path: use BFS
 - Edmonds Karp, runs in $O(VE^2)$
 - Even better:
 - Dinic Algorithm, runs in $O(V^2E)$
 - Faster in flow graph, where usually $|E| > |V|$ but $|E| \ll |V|^2$
 - Algorithm are not discussed today
 - [Code](#) is created by Su Zhan's friend in Fudan (Huang Jun)

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Solution: Max Flow (1)

- Solution for MWIS on Bipartite Graph
 - Max Flow:
 - Assign vertex cost as capacity from source (and to sink), respectively; give 'infinite' capacity in between $A \leftarrow B$



CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

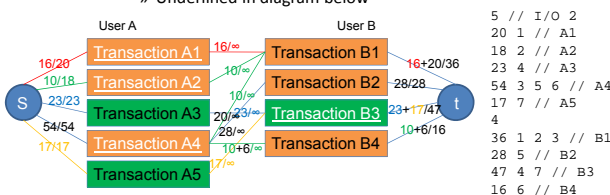
Summary about MWIS

- Summary:
 - DP for MWIS on **Tree** runs in $O(V)$
 - MWIS on **Bipartite Graph** depends on max flow algorithm used, it is $O(V^2E)$ using Dinic algorithm
 - Btw, Tree is also a bipartite graph...
 - Solution for MWIS on Bipartite Graph can also be used for MWIS on Tree, but slower (not a good choice...)
 - But MWIS on **general graph** is NP-Complete^... >.<
 - Usually not asked in contest...
 - Or if asked, input size is small for complete search algorithm

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Solution: Max Flow (2)

- Run $O(VE^2)$ Edmonds Karp on this Flow graph
 - Later, we will see $O(V^2E)$ Dinic algorithm
 - MWIS = |Total Weight| - Maxflow = 259 - 120 = 139
 - Solution: $\{s\text{-component} \cap \text{User A}\} + \{t\text{-component} \cap \text{User B}\}$
 - Underlined in diagram below



CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Special Graphs in Contest

- 3 special graphs frequently appear in contest:
 - Tree, keywords: connected, V vertices, V-1 edges!
 - Directed Acyclic Graph, keywords: no cycle
 - Bipartite, keywords: 2 sets, no edges within set!
 - Omitted in IOI
- Some classical 'hard' problems may have *polynomial solution* on these special graphs
 - This allows problem setter to increase input size!
 - Eliminates competitors who are not aware of the polynomial solution since the solution for general graph is typically 'intractable' (TLE)...

CS3233 - Competitive Programming, Semester 2, 2008/09, SoC, NUS

Directed Acyclic Graph (DAG)

- Some algorithms become simpler when used on DAGs instead of general graphs, based on the principle of topological ordering. For example, it is possible to find [shortest paths](#) and [longest paths](#) from a given starting vertex in DAGs in **linear time** by processing the vertices in a **topological order**, and calculating the path length for each vertex to be the minimum or maximum length obtained via any of its incoming edges. In contrast, for arbitrary graphs the shortest path may require slower algorithms such as [Dijkstra's algorithm](#) or the [Bellman-Ford algorithm](#), and longest paths in arbitrary graphs are [NP-hard](#) to find.

CS3233 - Competitive Programming,
Semester 2, 2009/10, SoC, NUS

References

- Introduction to Algorithms, p643-698, Ch 26
- Algorithm Design, p337-450, Ch 7
- Algorithms (Dasgupta et al), Ch 6 & Ch 7
- Algorithms (Sedgewick), Ch 33 & Ch 34
- Algorithms (Alsuwaiyel), Ch 16 & Ch 17
- Programming Challenges, p227-230, Ch 10
- Internet: [TopCoder Max-Flow tutorial](#), UVA, other Max Flow lecture notes, etc...

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Discuss CS3233 Contest 5 Prob B

- Solutions?
 - DAG?? Where is the DAG??
 - Longest path on DAG
 - Binary search the answer
- Also remember Banditji in APIO 09
 - Anyway SCC on DAG is outside IOI syllabus

CS3233 - Competitive Programming,
Semester 2, 2009/10, SoC, NUS

BE A PROBLEM SETTER

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Summary of Lecture

- You can identify *keywords* in problem wording
 - May be it can be modeled into known problem
- Some *'hard'* graph problems have polynomial solution when the graph is *'special'*
 - In IOI, this means: tree related algorithm or directed acyclic graph
 - So, learn them!
 - There are more than what discussed here

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Be a Problem Setter

- | | |
|---|---|
| <ul style="list-style-type: none"> • Problem Solver: <ol style="list-style-type: none"> Read the problem Think of a good algorithm Write 'solution' Create tricky I/O If WA, go to A/B/C/D If TLE/MLE, go to A/B/C/D If AC, stop ☺ | <ul style="list-style-type: none"> • Problem Setter: <ol style="list-style-type: none"> Write a <u>good</u> problem Write a <u>good</u> solution Set a <u>good</u> secret I/O Set problem settings • A problem setter <u>must</u> think from a different angle! <ul style="list-style-type: none"> – By setting good problems, you will simultaneously be a better problem solver!! |
|---|---|

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Problem Setter Tasks (1)

- Write a good problem
 - Options:
 - Pick an algorithm → find problem/story, or
 - Find a problem/story → identify a good algorithm for it (harder)
 - Problem description must not be ambiguous
 - Specify input constraints
 - Good English!
 - Easy: long, Hard: short!
- Write a good solution
 - Must be able to solve your own problem!
 - To set hard problem, one must increase his own programming skill!
 - Use the best possible algorithm with lowest time complexity
 - Not just the one 'that works'...

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Problem Setter Tasks (2)

- Set a good secret I/O
 - Tricky test cases to check AC vs WA
 - Usually 'boundary case'
 - Large test cases to check AC vs TLE/MLE
 - Perhaps use input generator to generate large test case, then pass this large input to our correct solution
- Set problem settings
 - Time Limit:
 - Usually 2 or 3 times the timings of your own best solutions
 - Java slower than C++!
 - Memory Limit:
 - Check OJ setting^
 - Problem Name:
 - Avoid revealing the algorithm in the problem name

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS

Be A Contest Organizer

- Contest Organizer Tasks:
 - Set problems of *various* topic
 - Better set by >1 problem setter
 - Must balance the difficulty of the problem set
 - Try to make it fun
 - Each team solves some problems
 - Each problem is solved by some teams
 - No team solve all problems
 - Every teams must work until the end of contest

CS3233 - Competitive Programming,
Semester 2, 2008/09, SoC, NUS