Tunnels Problem

Programming Puzzles and Competitions
CIS 4900 / 5920
Spring 2009

Outline

- A problem from ACM ICPC'07
- An example of a Min Cut problem
- Also uses an adaptation of the dynamic programming scheme of the Floyd-Warshall algorithm

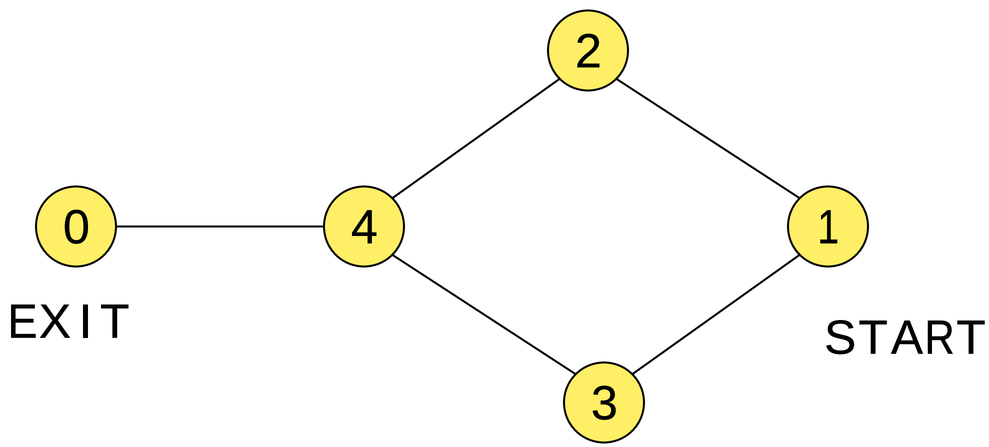
Tunnels Problem

- Spy is inside underground complex.
- Rooms are connected by point-to-point tunnels, plus tunnels to the outside.
- Spy wants to escape.
- We can track him and set off explosions that collapse tunnels.
- We need a strategy that will prevent him from escaping but destroy the smallest number of tunnels.

** From 2007 ACM ICPC contest*

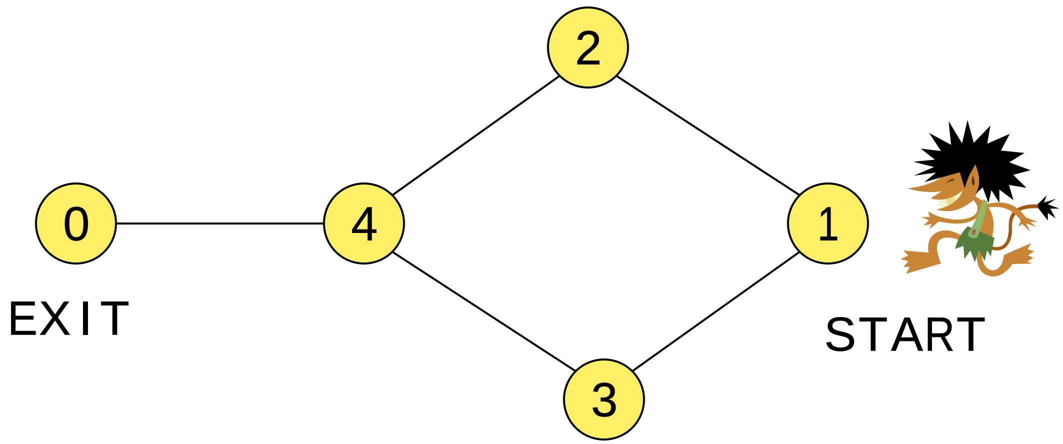
Example 1

4 6
1 2
1 3
2 4
3 4
4 0
4 0



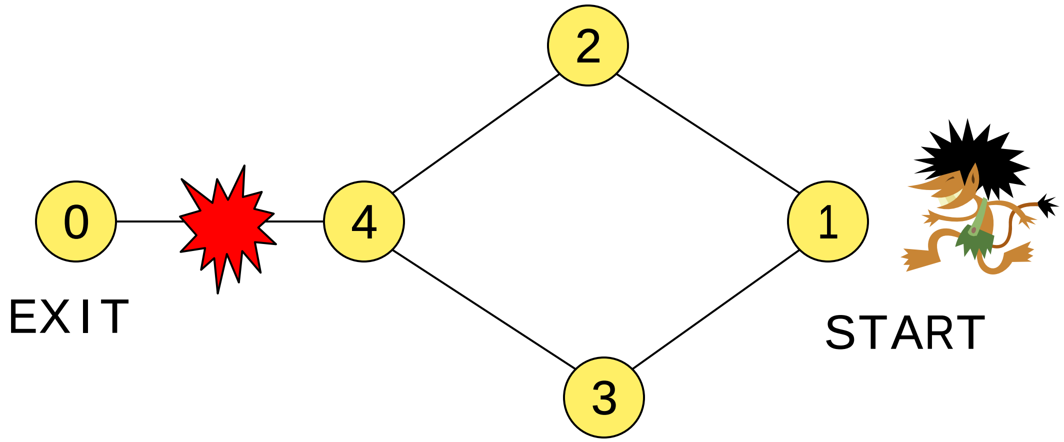
Example 1

4 6
1 2
1 3
2 4
3 4
4 0
4 0



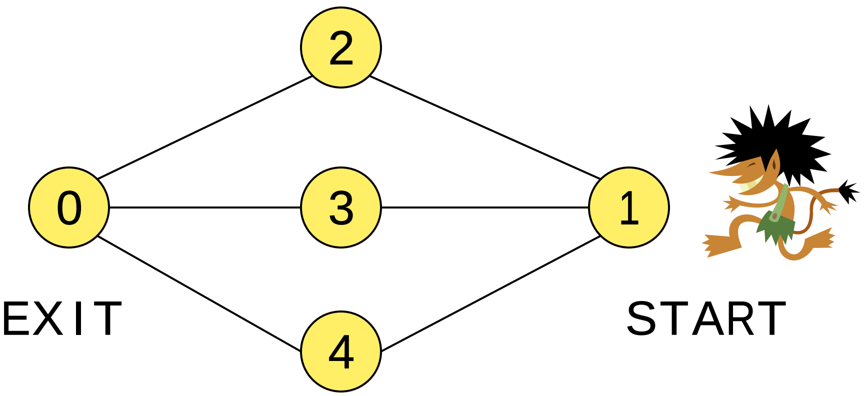
Example 1

4 6
1 2
1 3
2 4
3 4
4 0
4 0



Example 2

4 6
1 2
1 3
1 4
2 0
3 0
4 0



MinCut Problem

Cut = partition of a graph into two parts: S contains the designated “source” node s and T contains the “target” node t.

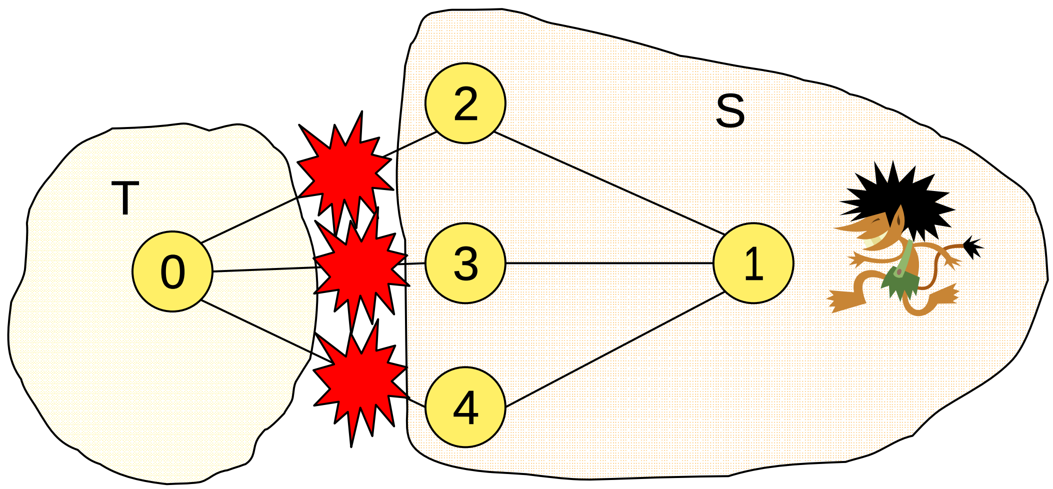
Cut size

$$c(S,T) = \sum_{u\in S, v\in T \,|\, (u,v)\in E} c(u,v)$$

Min Cut = minimum-size cut with given source and target nodes.

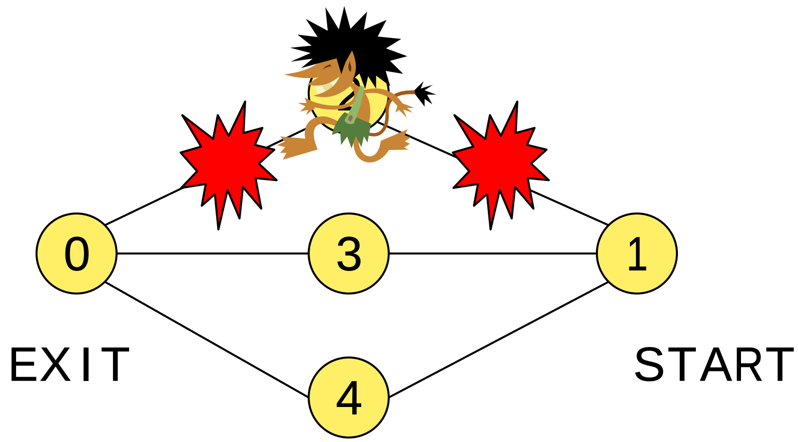
Is this a min-cut problem?

4 6
1 2
1 3
1 4
2 0
3 0
4 0



No, it’s not quite that simple

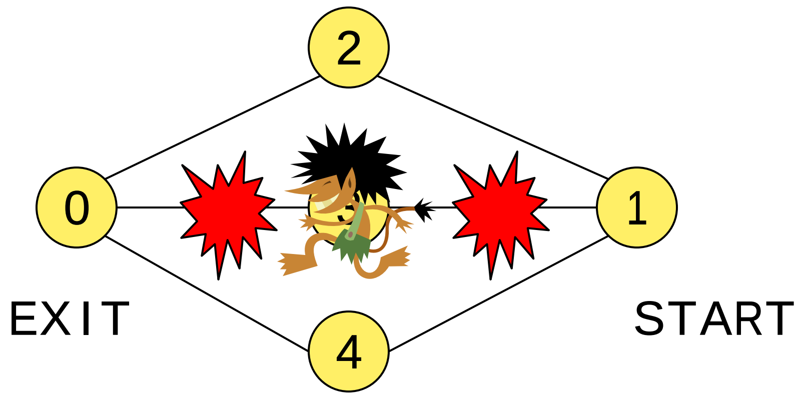
4 6
1 2
1 3
1 4
2 0
3 0
4 0



but this <u>is</u> a min-cut with source 2

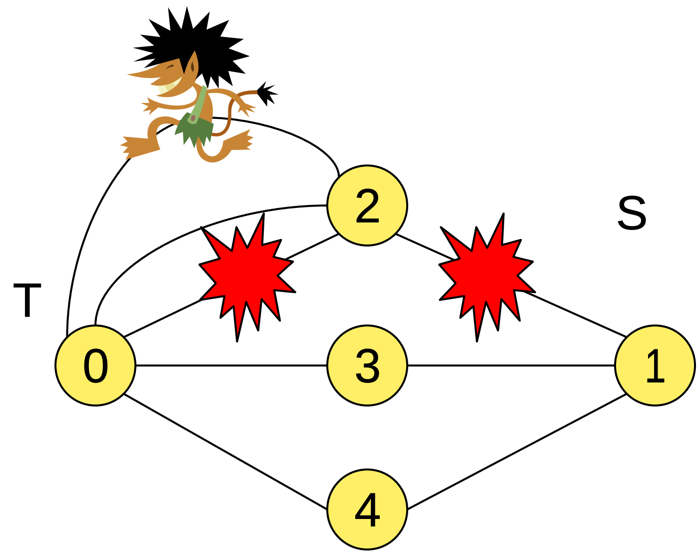
We need to consider all sources that lie between 1 and 0

4 6
1 2
1 3
1 4
2 0
3 0
4 0



Need to consider duplicate edges

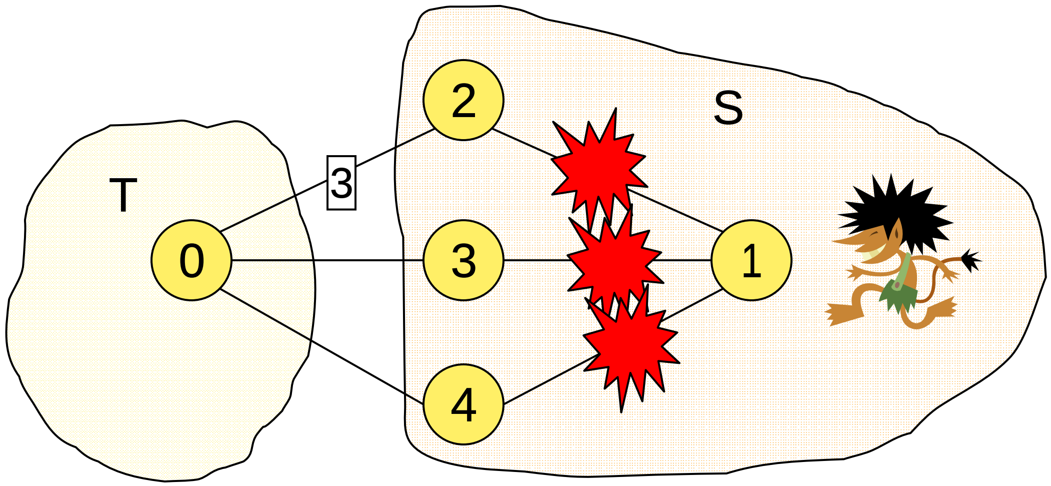
4 6
1 2
1 3
1 4
2 0
2 0
3 0
4 0



Lesson: Read the description carefully. Don't rely on the provided test cases.

Use integer-weighted edges

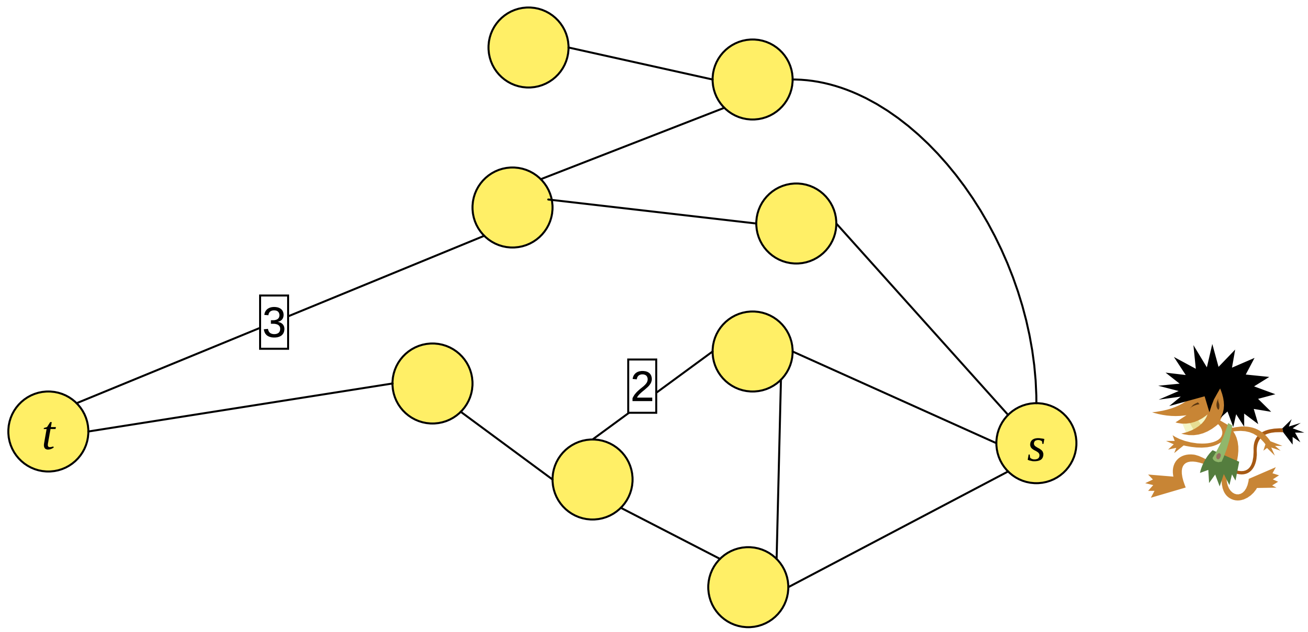
4 6
1 2
1 3
1 4
2 0
2 0
3 0
4 0



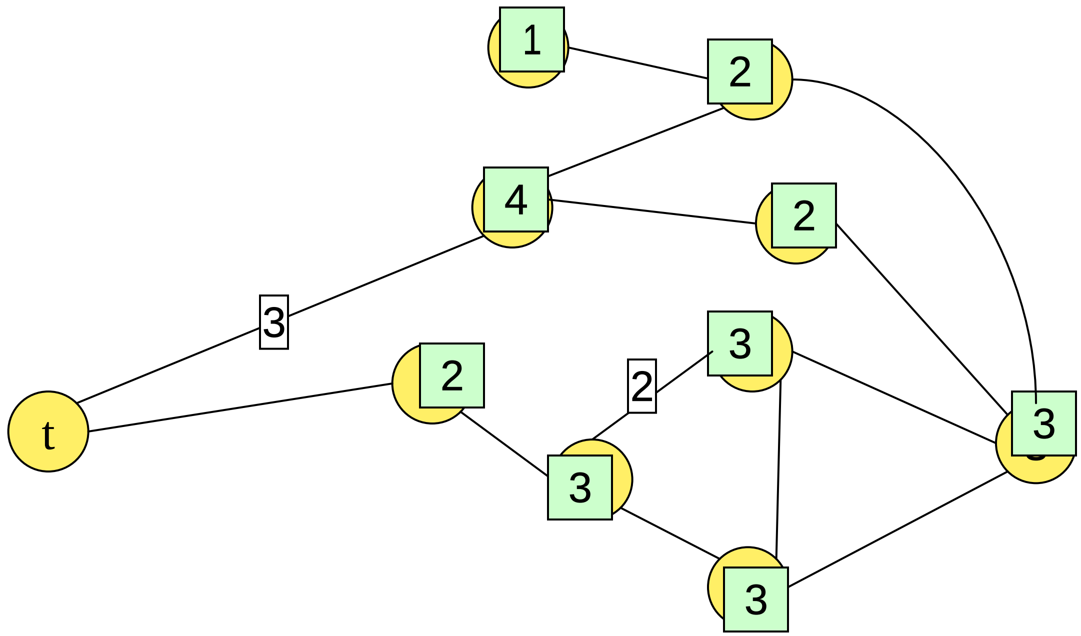
Strategy

1. Compute mincut size $minCut(x)$ for every source node x.
2. Compute $maxMinMinCut(s,t)$ as the maximum, over all paths p from s to t, of the minimum value of $minCut(v)$ over all the nodes v along the path p.
3. If $minCut(s) \leq maxMinMinCut(s,t)$, blow up $minCut(s)$ tunnels and block the spy where he is.
4. Otherwise, delay the decision and when the spy moves to another room, repeat from 2 with the new room as s.

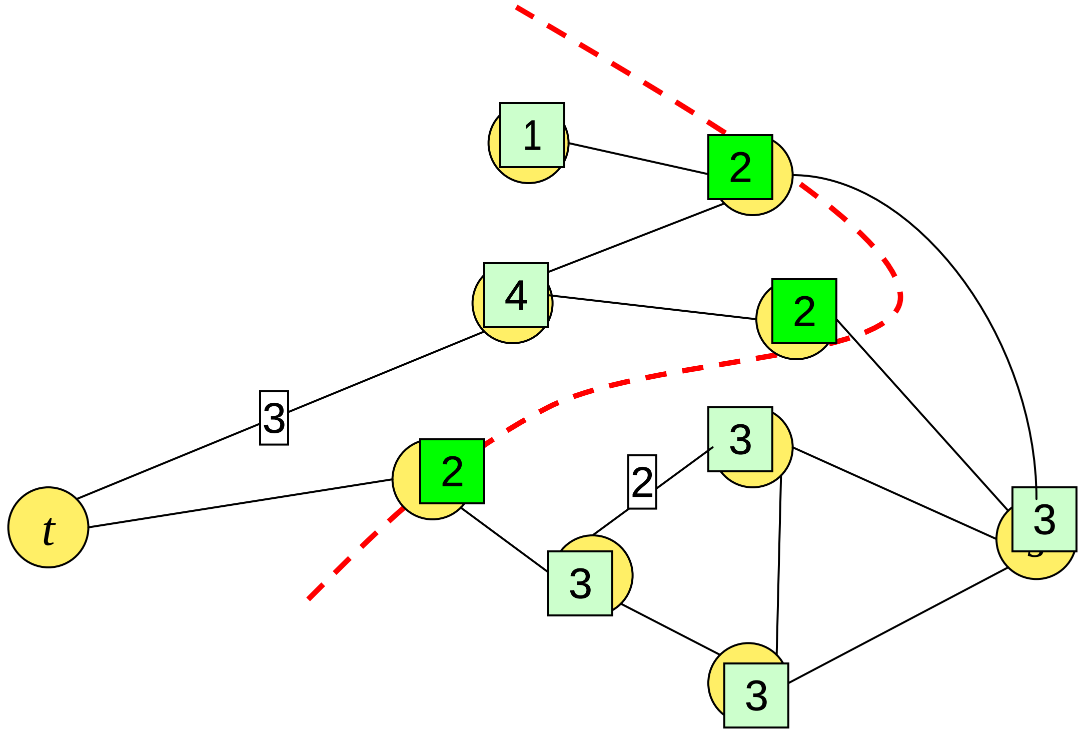
Example



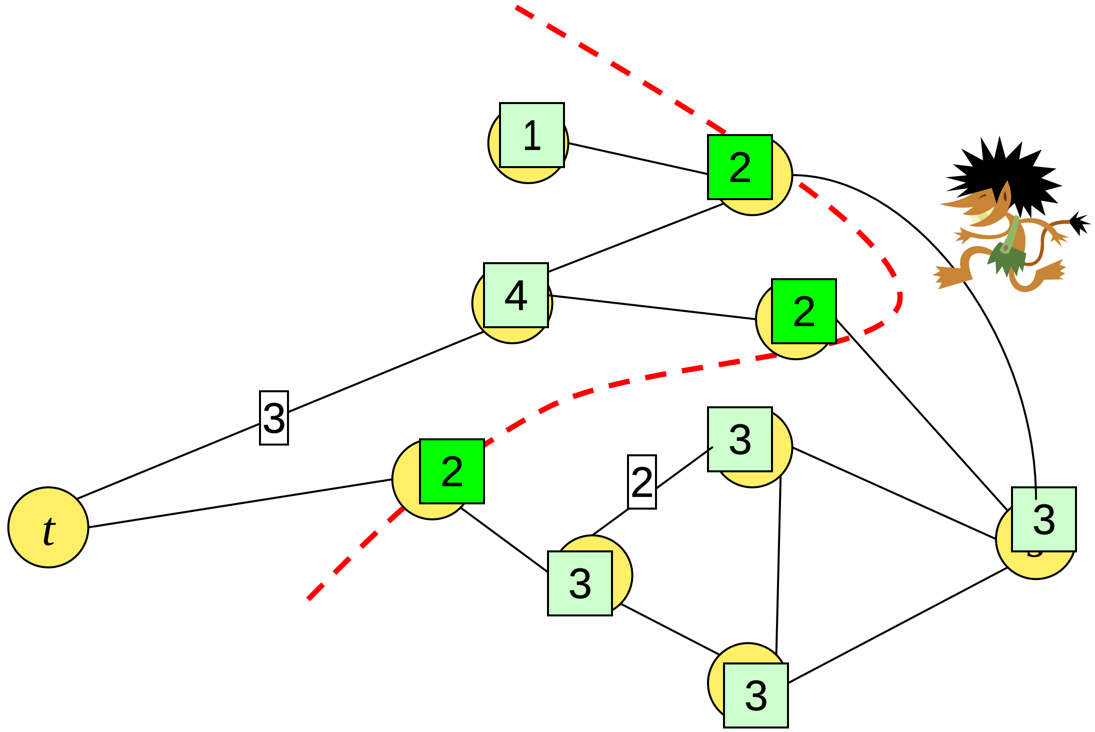
All-sources MinCut



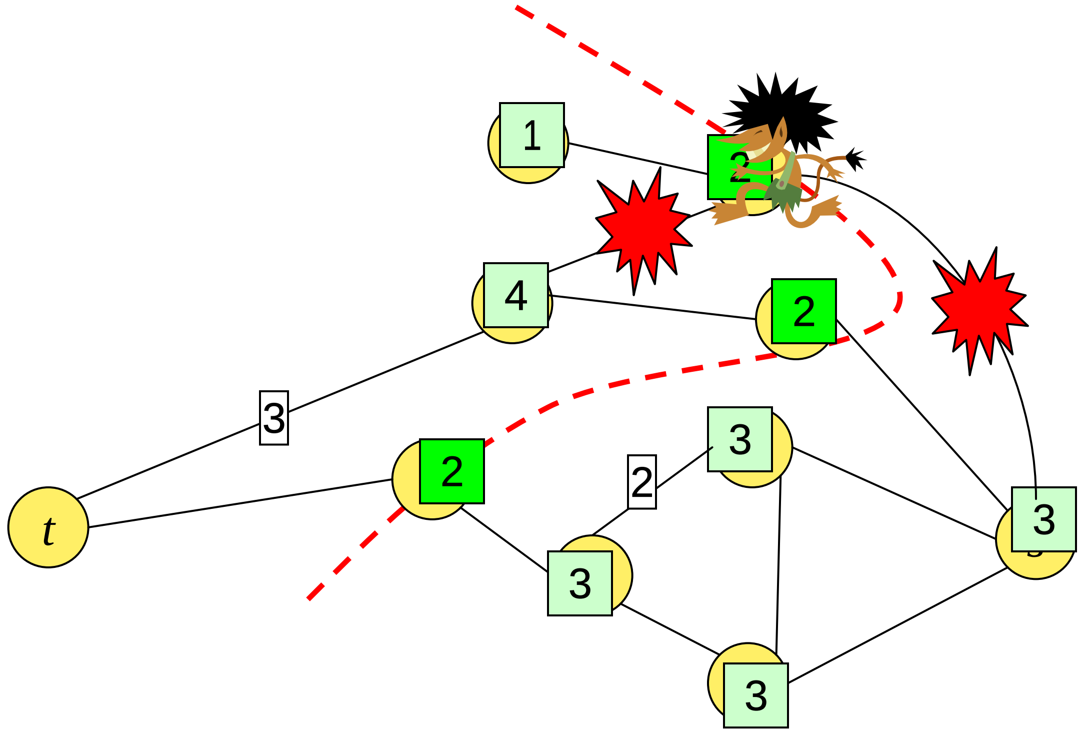
maxMinMinCost (s,t) = 2



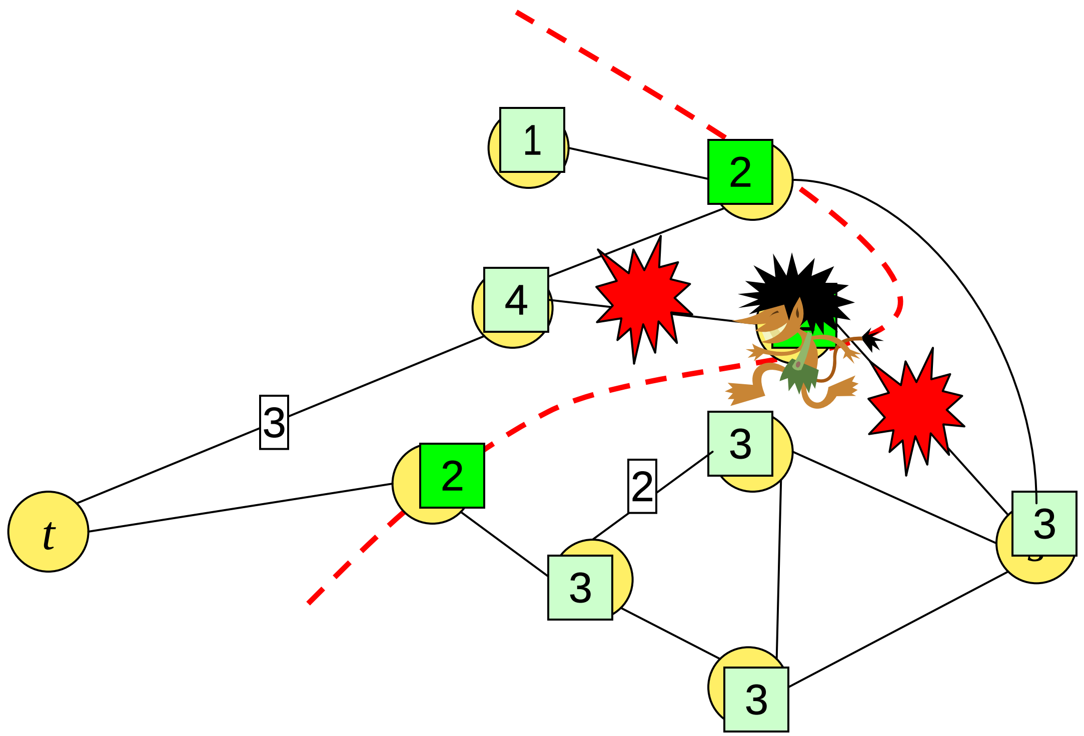
Application of the strategy



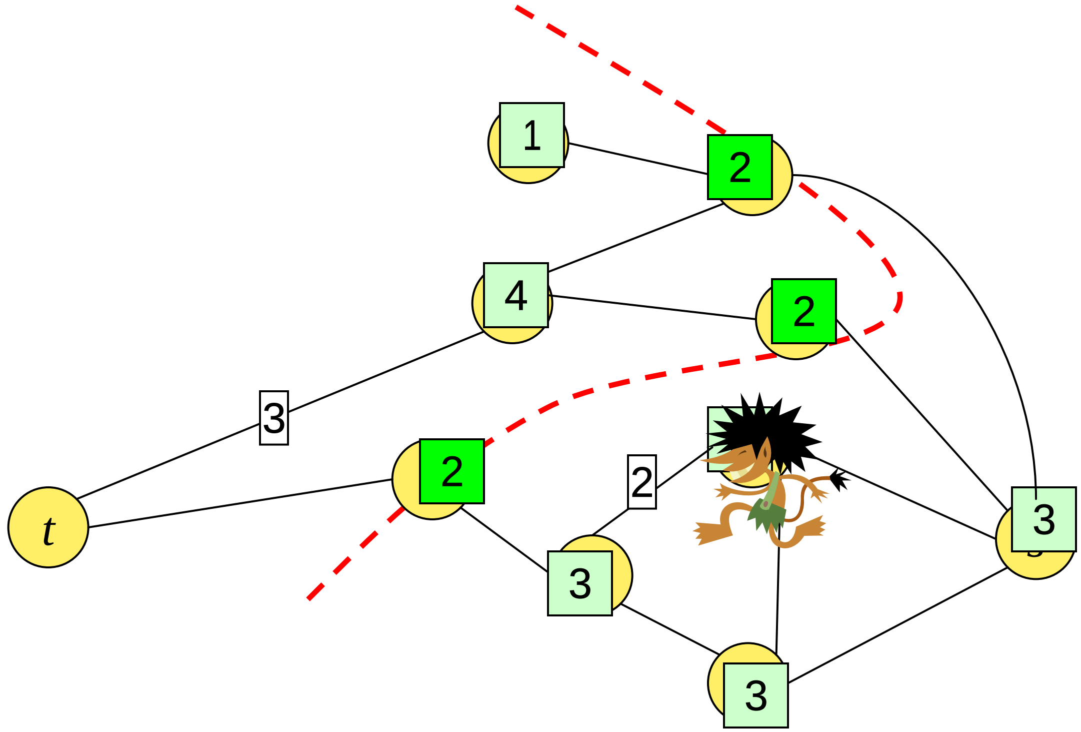
Application of the strategy



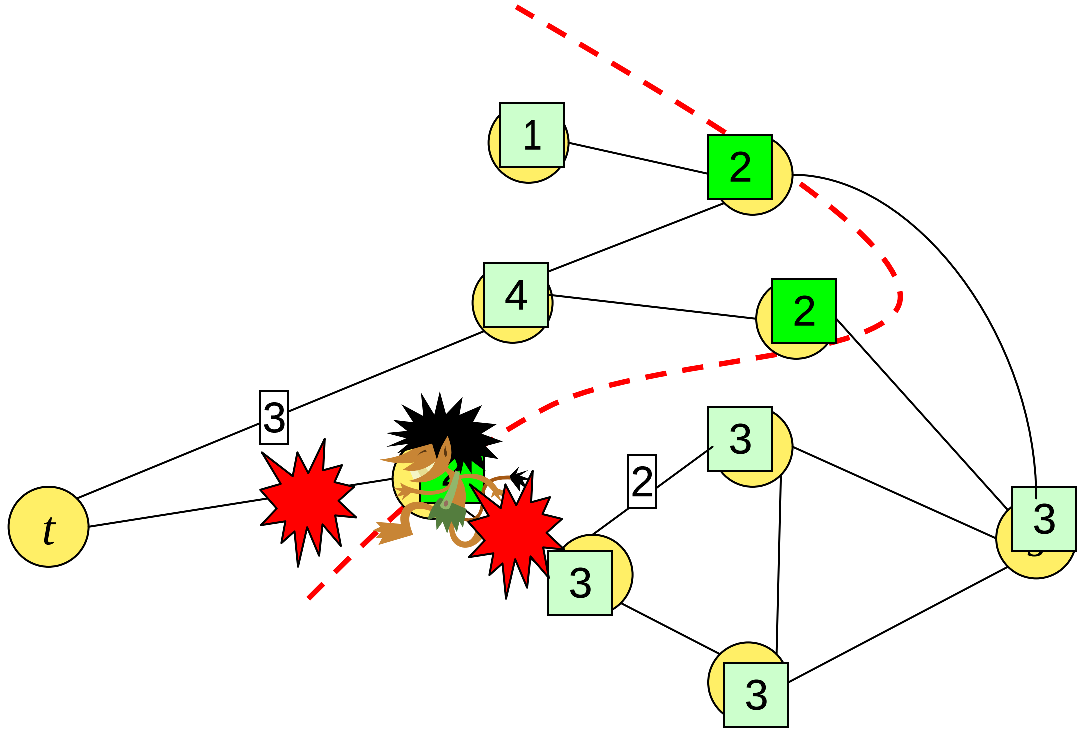
Another case



Another case



Another case, continued

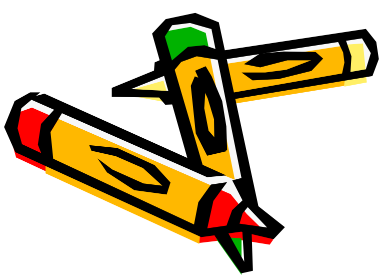


Review:

How to compute minCost(s)?

Ford-Fulkerson Algorithm

- Start with zero flow
- Repeat until convergence:
  - Find an augmenting path, from s to t along which we can push more flow
  - Augment flow along this path



See separate notes on this algorithm.

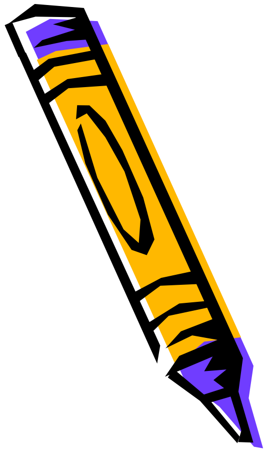
Ford-Fulkerson Algorithm

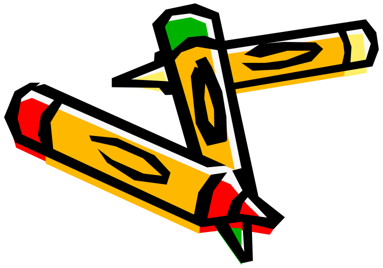
for (each edge (u,v) ϵ E[G])
  f[u][v] = f[v][u] = 0; } $\mathcal{O}$(E)
while ($\exists$ path p from s to t in G_f) { } $\mathcal{O}$(E)
  c_f(p) = min {c_f(u,v) | (u,v) ϵ p};
  for (each edge (u,v) ϵ p) {
    f[u][v] = f[u][v] + c_f(p)
    f[v][u] = -f[u][v]
  }
}

$\mathcal{O}$(E x f*) (for the whole while loop)

f = maximum flow, assuming integer flows, since each iteration increases flow by at least one unit*

```
int findMaxFlow (int s, int t) {
    int result = 0;
    for (int i = 0; i < n; i++)
        for (int j  = 0; j < n; j++)   flow[i][j] = 0;
    for (;;) {
        int Increment = findAugmentingPath(s, t);
        if (Increment == 0)  return result;
        result += capTo[t];
        int v = t, u;
        while (v != s) { // augment flow along path
            u = prev[v];
            flow[u][v] += capTo[t];
            flow[v][u] -= capTo[t];
            v = u;
}}}
```





```
static int findAugmentingPath(int s, int t) {
   for (int i = 0; i < n; i++) {
         prev[i] = -1;
         capTo[i]  = Integer.MAX_VALUE;}
   int first = 0, last = 0;
   queue[last++] = s; prev[s] = -2;   // s visited already
   while (first != last) {
         int u = queue[last--];
         for (int v = 0; v < n; v++) {
            if (a[u][v] > 0) {
               int edgeCap = a[u][v] - flow[u][v];
               if ((prev[v] == -1) && (edgeCap > 0)) {
                  capTo[v] = Math.min(capTo[u], edgeCap);
                  prev[v] = u;
                  if (v == t) return capTo[v];
                  queue[last++] = v;
    }}}}
   return 0;
}
```

This uses depth-first search.

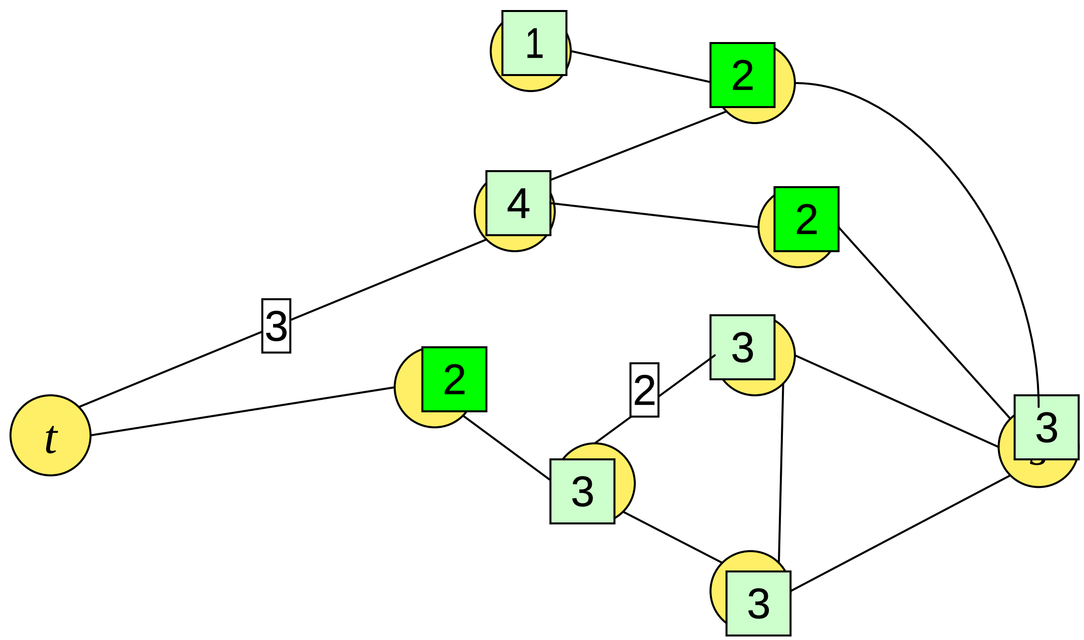
Next:

How to compute maxMinMinCost(u,v)?

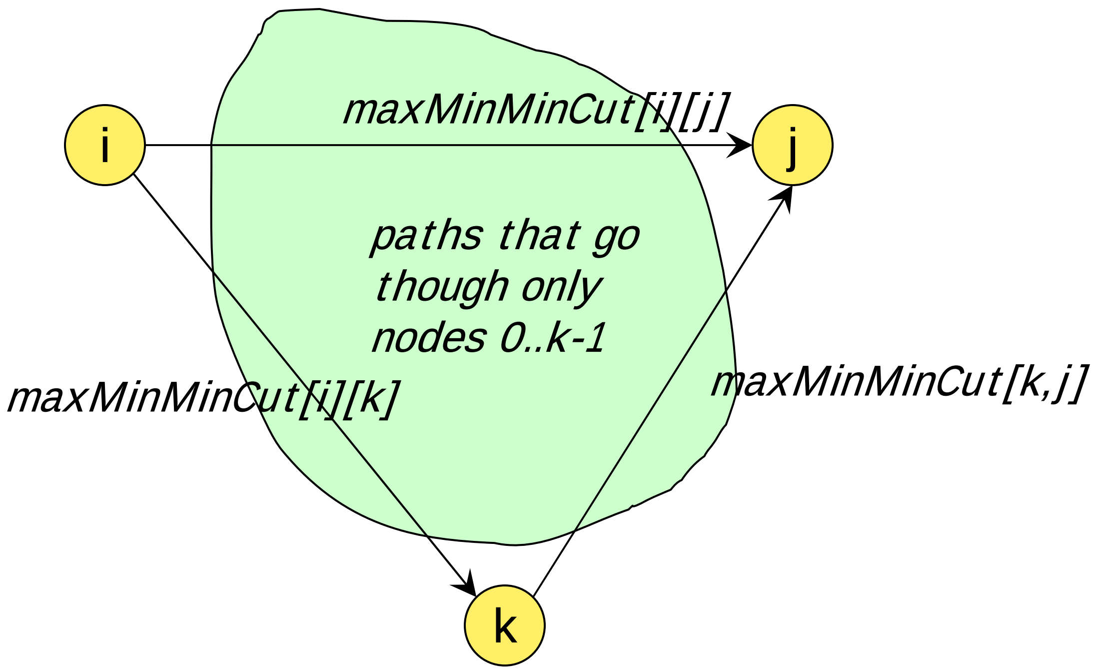
minCut[p] = min {minCut[v] : v ϵ p, v ≠ t}

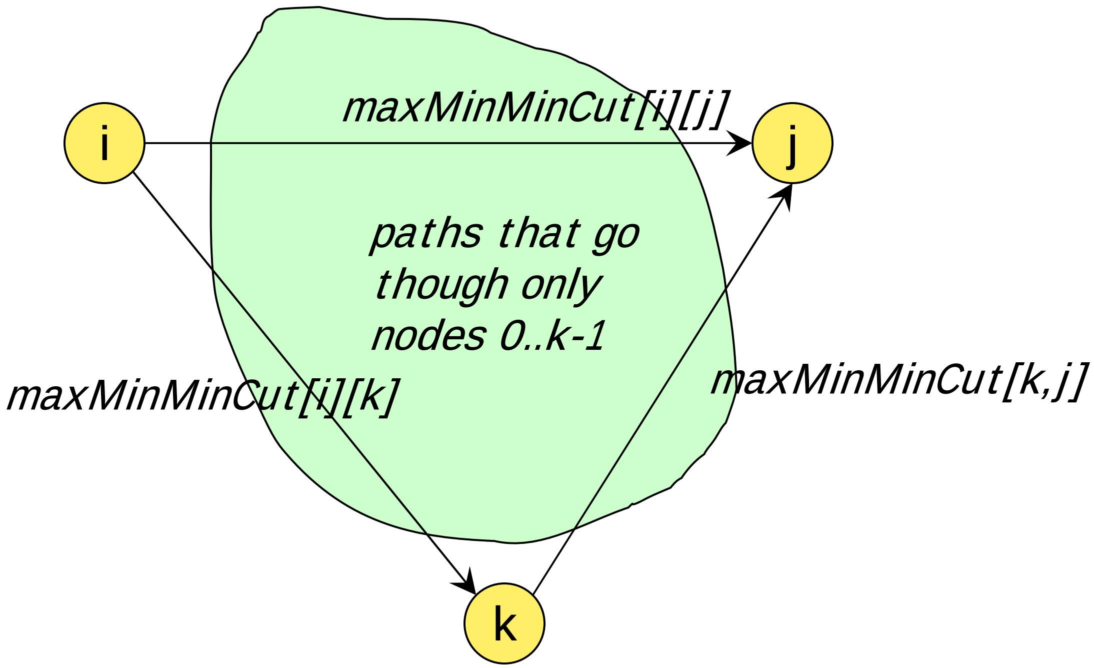
maxMinMinCut[i][j] = x iff

1. ∃ path p from i to j such that minCut[p] = x, and
2. ∀ path p from i to j minCut[p] ≤ x



Analogy to shortest path





for k=0, maxMinMinCut[i,j] = minCut[j].

for k>0 maxMinMinCut[i,j] = minCut[j].

```
static void findMaxMinMinCut() {
    int ij, ik, kj, kk;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (a[i][j] > 0) maxMinMinCut[i][j] = minCut[j];
    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) {
                ij = maxMinMinCut[i][j];
                ik = maxMinMinCut[i][k];
                kj = maxMinMinCut[k][j];
                if ((ik > 0) && (kj > 0)) {
                    kk = Math.min (ik, kj);
                    if (kk > ij) maxMinMinCut[i][j] = kk;
                }
            }
}
```

This uses depth-first search.

Full program:

www.cs.fsu.edu/~baker/pc/tunnels/Tunnels.java