Two Different Approaches

Prim’s Algorithm
Almost identical to Dijkstra’s

Kruskals’s Algorithm
Completely different!

Kruskal’s MST Algorithm

Idea: Grow a forest out of edges that do not create a cycle. Pick an edge with the smallest weight.

G=(V,E)

Kruskal’s Algorithm for MST

An edge-based greedy algorithm
Builds MST by greedily adding edges

1. Initialize with
   • empty MST
   • all vertices marked unconnected
   • all edges unmarked
2. While there are still unmarked edges
   a. Pick the lowest cost edge \((u, v)\) and mark it
   b. If \(u\) and \(v\) are not already connected, add \((u, v)\) to the MST and mark \(u\) and \(v\) as connected to each other

Doesn’t it sound familiar?

Kruskal Pseudo Code

```cpp
void Graph::kruskal()
{
    int edgesAccepted = 0;
    DisjSet s(NUM_VERTICES);
    while (edgesAccepted < NUM_VERTICES – 1)
    {
        e = smallest weight edge not deleted yet;
        // edge e = (u, v)
        uset = s.find(u);
        vset = s.find(v);
        if (uset != vset)
        {
            edgesAccepted++;
            s.unionSets(uset, vset);
        }
    }
}
```

Complexity?

Find MST using Kruskal’s

```
A 2
B 2
C 3
D 1
```

Total Cost:

- Now find the MST using Prim’s method.
- Under what conditions will these methods give the same result?

Kruskal’s Algorithm: Correctness

It clearly generates a spanning tree. Call it \(T_K\).

Suppose \(T_K\) is not minimum:

Pick another spanning tree \(T_{ext}\) with lower cost than \(T_K\)
Pick the smallest edge \(e_s(u,v)\) in \(T_K\) that is not in \(T_{ext}\)
\(T_{ext}\) already has a path \(p\) in \(T_{ext}\) from \(u\) to \(v\)
⇒ Adding \(e_s\) to \(T_{ext}\) will create a cycle in \(T_{ext}\)
Pick an edge \(e_e\) in \(p\) that Kruskal’s algorithm considered after adding \(e_s\) (must exist: \(u\) and \(v\) unconnected when \(e_s\) considered)
⇒ \(\text{cost}(e_s) \geq \text{cost}(e_e)\)
⇒ can replace \(e_e\) with \(e_s\) in \(T_{ext}\) without increasing cost!
Keep doing this until \(T_{ext}\) is identical to \(T_K\)
⇒ \(T_K\) must also be minimal – contradiction!
Return to Dynamic Programming

• Recall that dynamic programming is a technique that reuses computed values of intermediate computations:
• $\text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2)$
• A classic description of a computation that is suitable for dynamic programming is the form $f(i, k) = \min_j (f(i, j) + f(j, k))$

Context Free Grammar

• A grammar $G=(T, N, S, P)$ where
  – $T$ is a set of terminals, e.g. $T=\{a, b, n, s\}$
  – $N$ is a set of non-terminals, e.g. $N=\{S, A, B\}$
  – $S$ is the start symbol
  – $P$ is a set of productions of the form $N \rightarrow \text{string}$

A Generation

$S \Rightarrow baA \Rightarrow banaA \Rightarrow bananaA \Rightarrow bananaB \Rightarrow bananas$

• Parse Tree

Alternative Grammar

• There are many ways to express strings by cfgs

Parse by Reversing Arrows

bananas $\Rightarrow$
$\Rightarrow$ bananas using $S \rightarrow b$
$\Rightarrow$ SBananas using $B \rightarrow a$
$\Rightarrow$ SBCanas using $C \rightarrow n$
$\Rightarrow$ SAanas using $A \rightarrow BC$
$\Rightarrow$ Sanas using $S \rightarrow SA$
$\Rightarrow$ SAnas using $A \rightarrow a$
$\Rightarrow$ Snas using $S \rightarrow SA$
$\Rightarrow$ ?

Parse By Dynamic Programming