CS3233
Competitive Programming

Dr. Steven Halim

Week 02 –
Data Structures & Libraries

Outline

- Mini Contest 1 + Break
- Data Structures With Built-in Libraries + Break
  - Linear Data Structures (CS1010/1st quarter of CS2020)
  - Non Linear Data Structures (CS2010/the remaining part of CS2020)
- Data Structures With Our-Own Libraries
  - Graph
  - Union-Find Disjoint Sets
  - Segment Tree
  - Fenwick Tree
- "Top Coder" Coding Style

Basic knowledge that all ICPC/IOI-ers must have!

LINEAR DATA STRUCTURES WITH BUILT-IN LIBRARIES

Linear DS + Built-In Libraries (1)

1. Static Array, built-in support in C/C++/Java
2. Resize-able: C++ STL <vector>, Java Vector
   - Both are very useful in ICPCs/IOIs

- There are 2 very common operations on Array:
  - Sorting
  - Searching
  - Let’s take a look at efficient ways to do them

One of the “fundamental” CS problem

SORTING OUR DATA

Sorting (1)

- Definition:
  - Given unsorted stuffs, sort them… *
- Popular Sorting Algorithms
  - $O(n^2)$ algorithms: Bubble/Selection/Insertion Sort
  - $O(n \log n)$ algorithms: Merge/Quick/Heap Sort
  - Special purpose: Counting/Radix/Bucket Sort
- Reference:
  - http://en.wikipedia.org/wiki/Sorting_algorithm

Sorting (2)

- In ICPC, you can "forget" all these…
  - In general, if you need to sort something…, just use the O(n log n) sorting library:
    - C++ STL <algorithm>: sort
    - Java: Collections.sort (not discussed in this lecture)
      - Java users: please study sample codes on your own
- In ICPC, sorting is either used as *preliminary step* for more complex algorithm or to *beautify output*
  - Familiarity with sorting libraries is a must!

Sorting (3)

- Sorting routines in C++ STL <algorithm>
  - sort – a bug-free implementation of *introsort**
    - Can sort basic data types (ints, doubles, chars), Abstract Data Types (C++ class), multi-field sorting (≥ 2 criteria)
  - partial_sort – implementation of *heapsort*
    - Can do O(k log n) sorting, if we just need top-k sorted!
  - stable_sort
    - If you need to have the sorting ‘stable’, keys with same values appear in the same order as in input.

Sorting (4)

- ∃ sorting algorithms faster than O(n log n) e.g.
  - Counting sort (only for special cases^)
  - Counting sort demo:
    - http://users.cs.cf.ac.uk/C.L.Mumford/tristan/CountPage.html
    - Example codes provided
  - UVa: 11462 (Age Sort)!
- There are others (not discussed today):
  - Radix*
  - Bucket sort, etc

Another “fundamental” CS problem

SEARCHING OUR DATA

Searching in Array

- Two variants:
  - When the array is sorted versus not sorted
- Must do O(n) linear scan if not sorted - trivial
- Can use O(log n) binary search when sorted
  - PS: must run an O(n log n) sorting algorithm once
- Binary search is ‘tricky’ to code!
  - Instead, use C++ STL <algorithm>: lower_bound

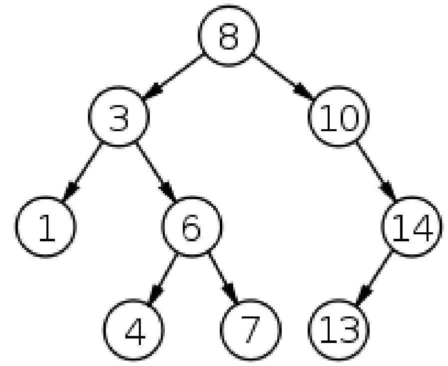
Linear DS + Built-In Libraries (2)

3. Linked List, C++ STL <list>, Java LinkedList
   – Usually not used in ICPCs/IOIs
4. Stack, C++ STL <stack>, Java Stack
   – Used by default in Recursion, Postfix Calculation
5. Queue, C++ STL <queue>, Java Queue
   – Used in Breadth First Search, Topological Sort, etc

More efficient data structures

NON-LINEAR DATA STRUCTURES WITH BUILT-IN LIBRARIES

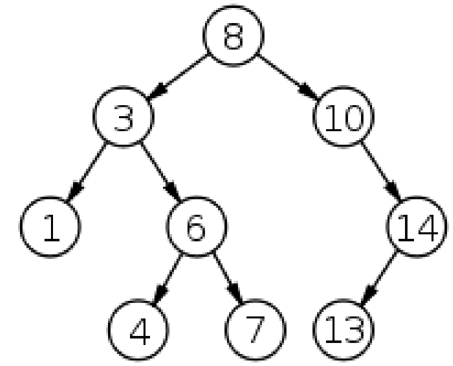
Binary Search Tree (1)



A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13

- ADT Table (key → data)
- Binary Search Tree (BST)
  – Advertised O(log n) for insert, search, and delete
  – Requirement: the BST must be **balanced**!
    - AVL tree, Red-Black Tree, etc… *argh*
- Do not worry, just use: C++ STL <map>
  – UVa 10295 (Hay Points)
  – UVa 10226 (Hardwood Species)*

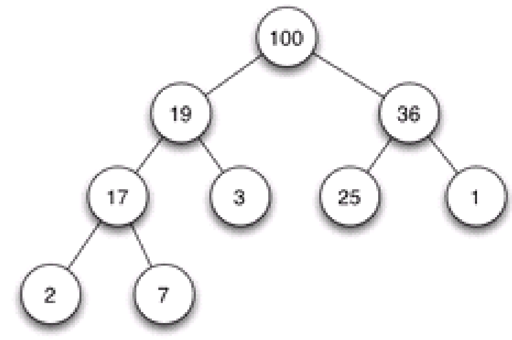
Binary Search Tree (2)



A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13

- ADT Table (key exists or not)
- Set (Single Set)
  - C++ STL <set>, similar to C++ STL <map>
    - <map> stores a pair<key, data>
    - <set> stores just the key
  - Can use <set> as vector with “auto-sort” feature

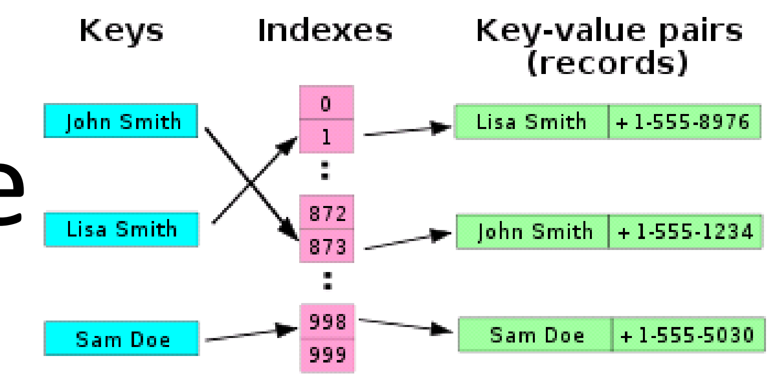
Heap



Example of a full binary max heap

- Heap
  - C++ STL <algorithm> has some heap algorithms
    - partial_sort uses heapsort
  - C++ STL <queue> has priority_queue (a heap)
    - Dijkstra and Kruskal's algorithms use priority queue
- But, we rarely see pure heap problems in ICPC
  - Perhaps for something like this:
    - Maintain top-k/bottom-k items
      given a very large stream of data…

Hash Table



A small phone book as a hash table.

- Hash Table
  - Advertised O(1) for insert, search, and delete, but:
    - The hash function must be good!
    - There is no Hash Table in C++ STL (∃ in Java API)
  - Nevertheless, O(log n) using <map> is usually ok
- Direct Addressing Table (DAT)
  - Rather than hashing, we more frequently use DAT
  - UVa [11340](11340) (Newspaper)
    - How about UVa [499](499)?

10 Minutes Break

- More data structures *without* built-in libraries will be discussed in the last part…
  - Many are outside CS1020/2010 syllabus
    - Graph (data structure only)
    - Union-Find Disjoint Sets
    - Segment Tree
    - Fenwick Tree
  - Some are discussed in CS2020 syllabus

Time Check:
8.10pm

Graph (data structure only)

Union-Find Disjoint Sets

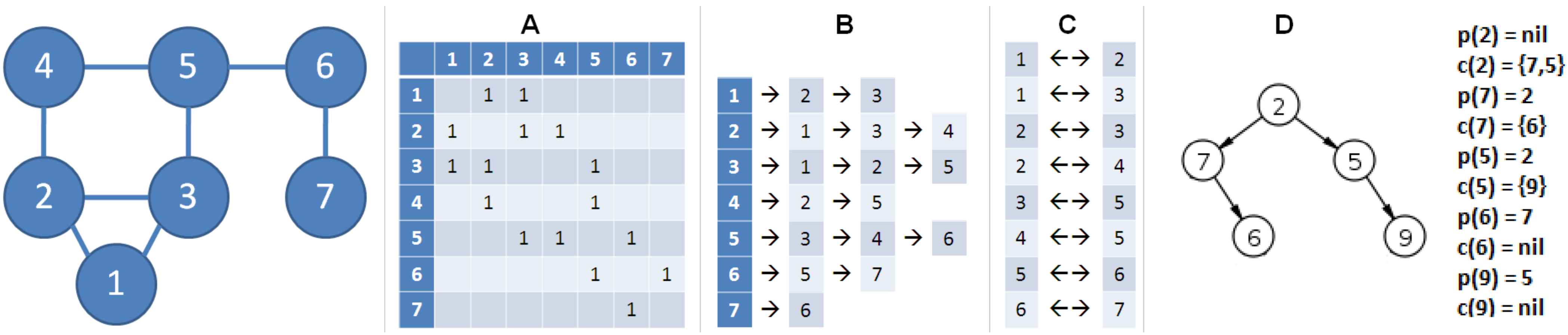
Segment Tree

Fenwick Tree

DATA STRUCTURES WITHOUT BUILT-IN LIBRARIES

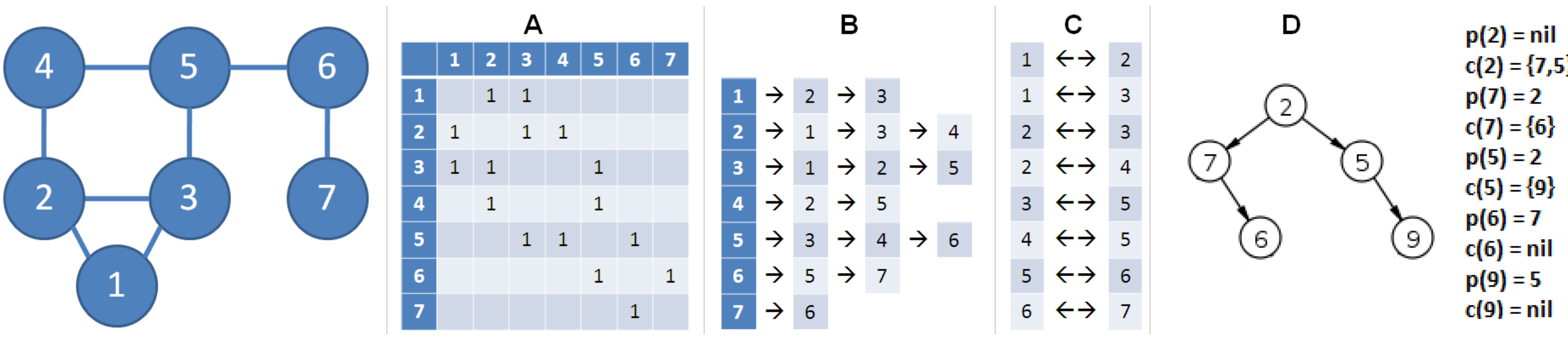
Graph Data Structures (1)

- Graph is a special data structure
  - Used to model objects and connections…
- Graph Representation:
  - int **AdjacencyMatrix**[V][V];
  - typedef pair<int, int> ii; typedef vector<ii> vii;
    vector<vii> **AdjacencyList**;



Graph Data Structures (2)

- Graph Representation:
  - typedef pair<int, int> ii;
    priority_queue<pair <int, ii > > **EdgeList**;
  - typedef vector<int> vi;
    int **parent**;
    vi **children**;



Graph Data Structures (3)

- Typical Input:

  3 // n
  0 2 3 // cell[i][j] > 0 implies that
  0 0 4 // ∃ an edge between vertex
  0 1 0 // i-j with weight cell[i][j]

- Adjacency Matrix:

```
#define MAX_N 10 // set size
int G[MAX_N][MAX_N], n;
scanf("%d", &n);
REP (i, 0, n - 1)
  REP (j, 0, n - 1)
    scanf("%d", &G[i][j]);
```

Undirected graph has
symmetric adjacency matrix

- Typical Input:

  3 // n
  2 1 2 2 3 // vertex 0 → 2 neighbors
  1 2 4 // pair (neighbor, weight)
  1 3 1

- Adjacency List (use STL):

```
vector<vii> G;
int V, t, j, w;
scanf("%d", &V);
REP (i, 0, V - 1) {
  vii Nb;
  scanf("%d", &t);
  while (t--) {
    scanf("%d %d", &j, &w);
    Nb.PB(ii(j, w));
  }
  G.PB(Nb);
}
```

Graph Data Structures (4)

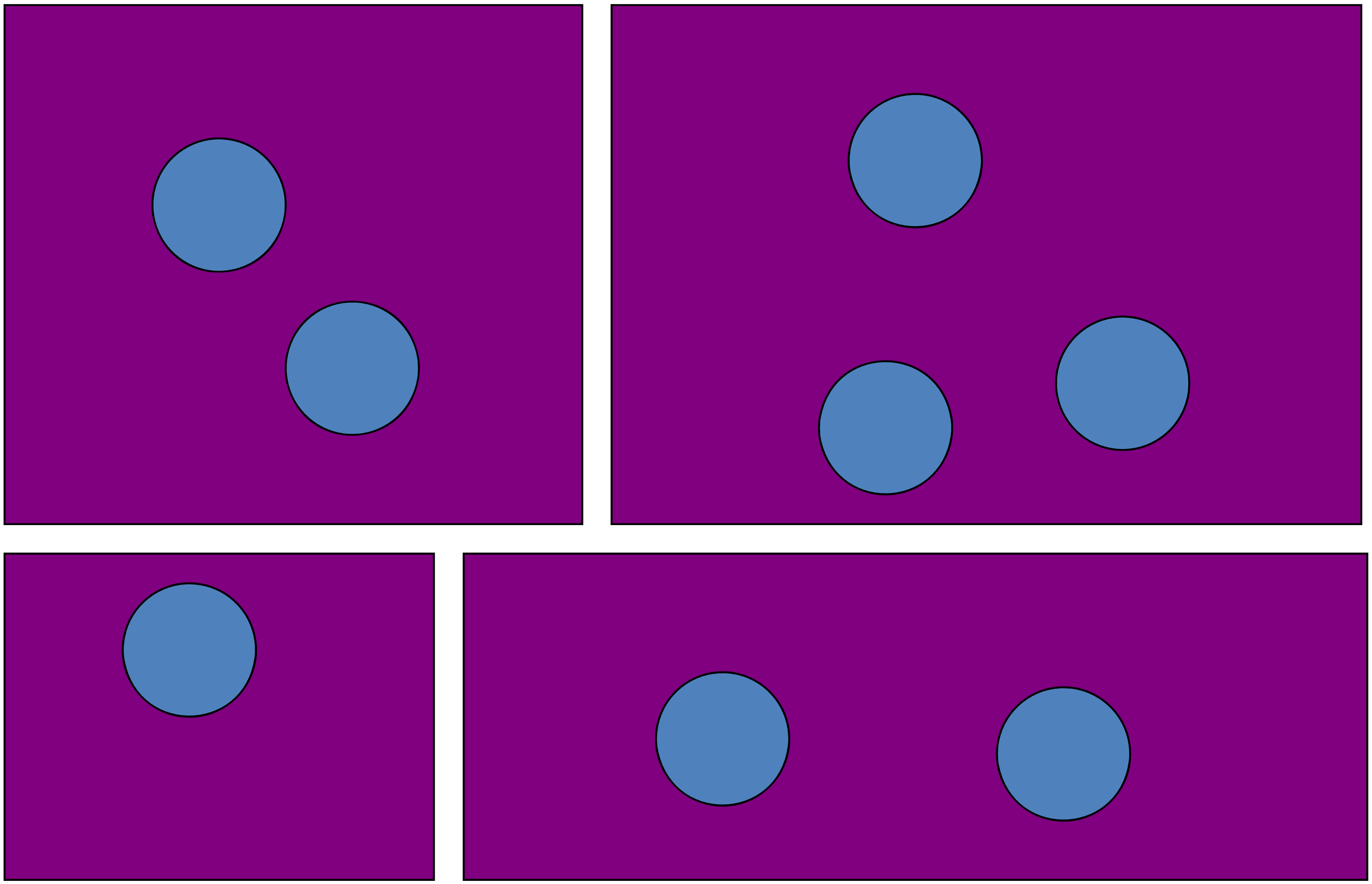
- Adjacency Matrix:
- Pro:
  - Existence of edge i-j can be found in O(1)
  - Good for dense graph/ Floyd Warshall's*
- Cons:
  - O(V) to enumerate neighbors of a vertex
  - $O(V^2)$ space

- Adjacency List:
- Pro:
  - O(k) to enumerate k neighbors of a vertex
  - Good for sparse graph/ Dijkstra's*/DFS/BFS
- Cons:
  - O(k) to check the existence of edge i-j

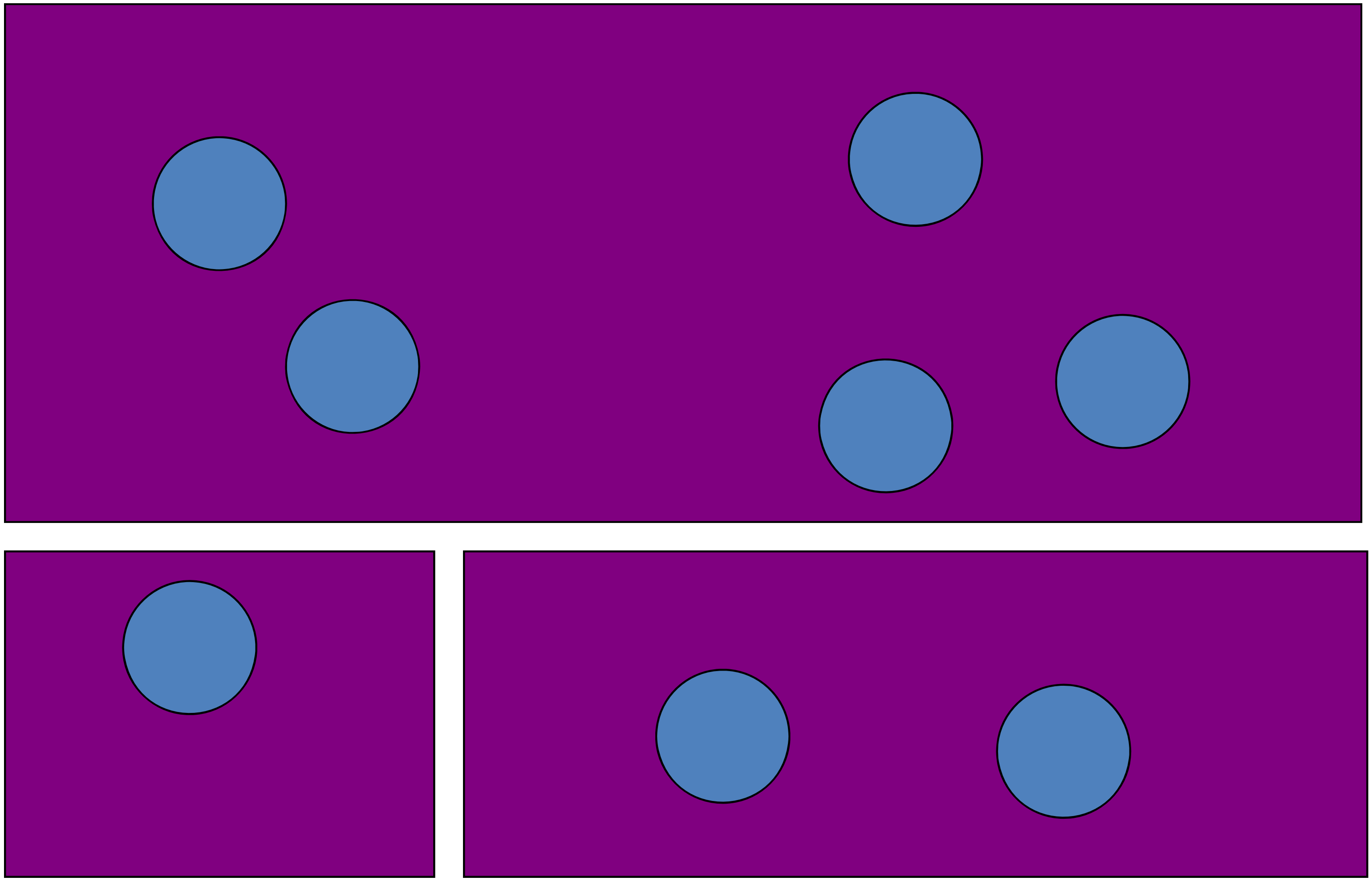
Union-Find Disjoint Sets (1)

- Disjoint-Set DS (Union Find)
  - Given several disjoint sets initially…
  - **Combine** them when needed!
  - UVa:
    - 459 (Graph Connectivity)
    - 793 (Network Connections)
    - 10608 (Friends)
    - 11503 (Virtual Friends)

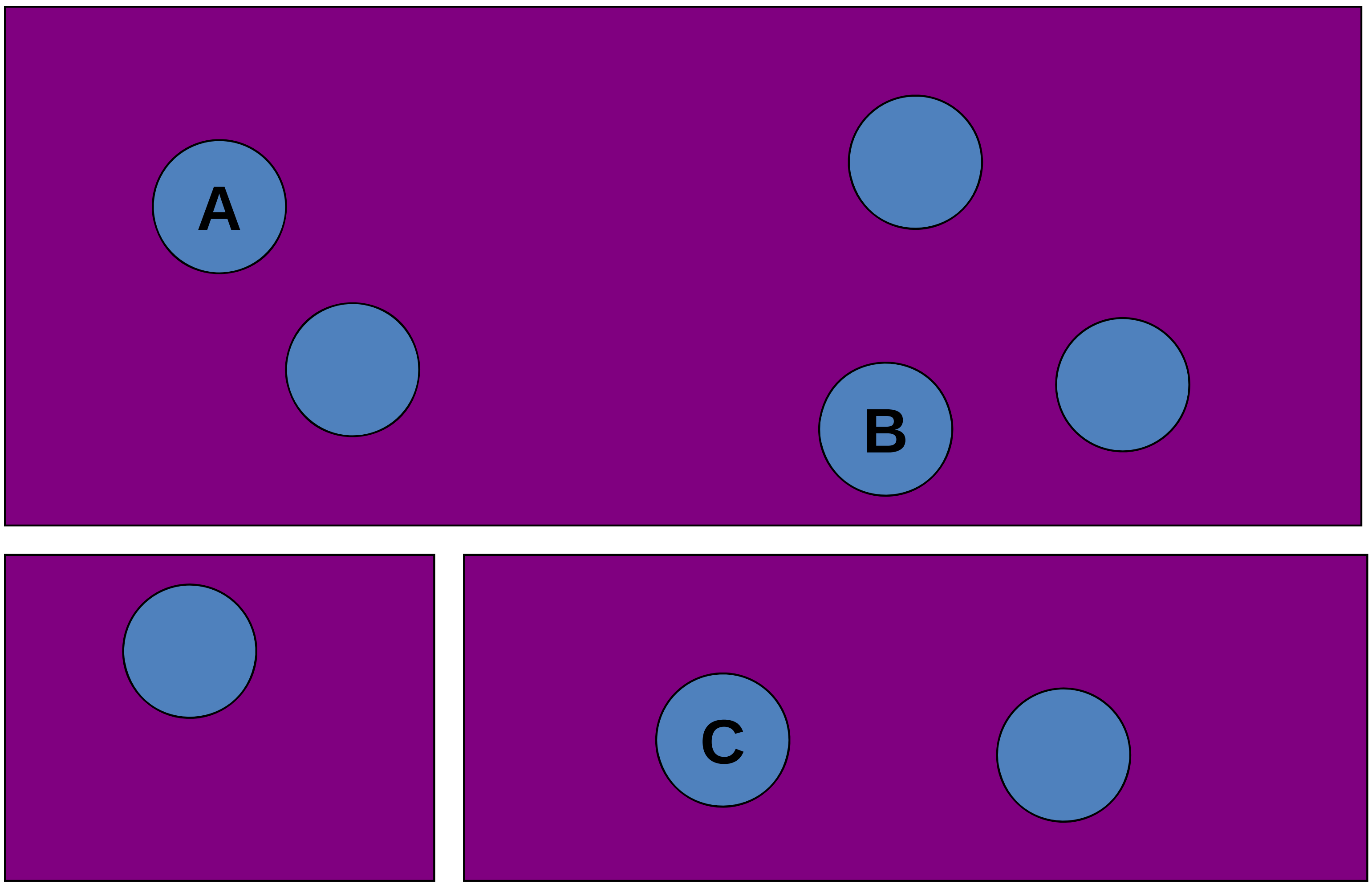
Overview



Operation Union



Operation Find



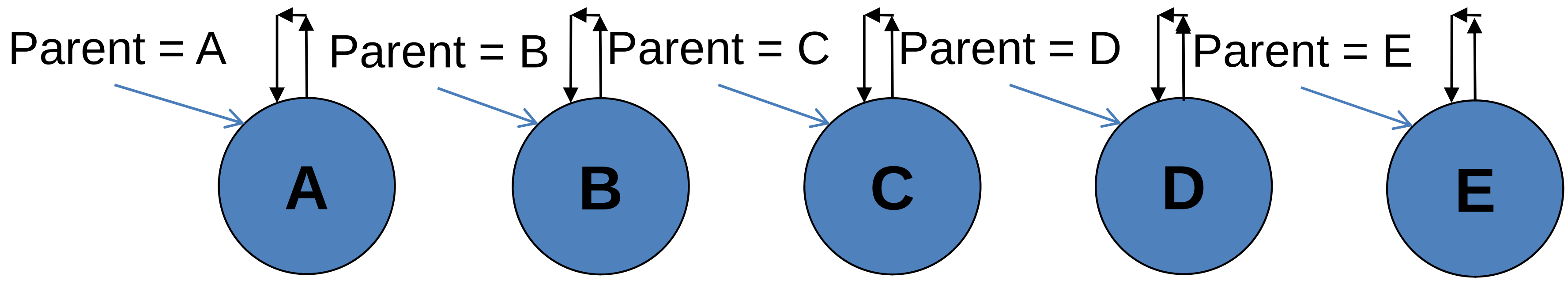
Applications

- Kruskal Minimum Spanning Tree algorithm
- Finding Connected Components in Graph
- Both discussed later in Week05-06
- etc

initSet(5)

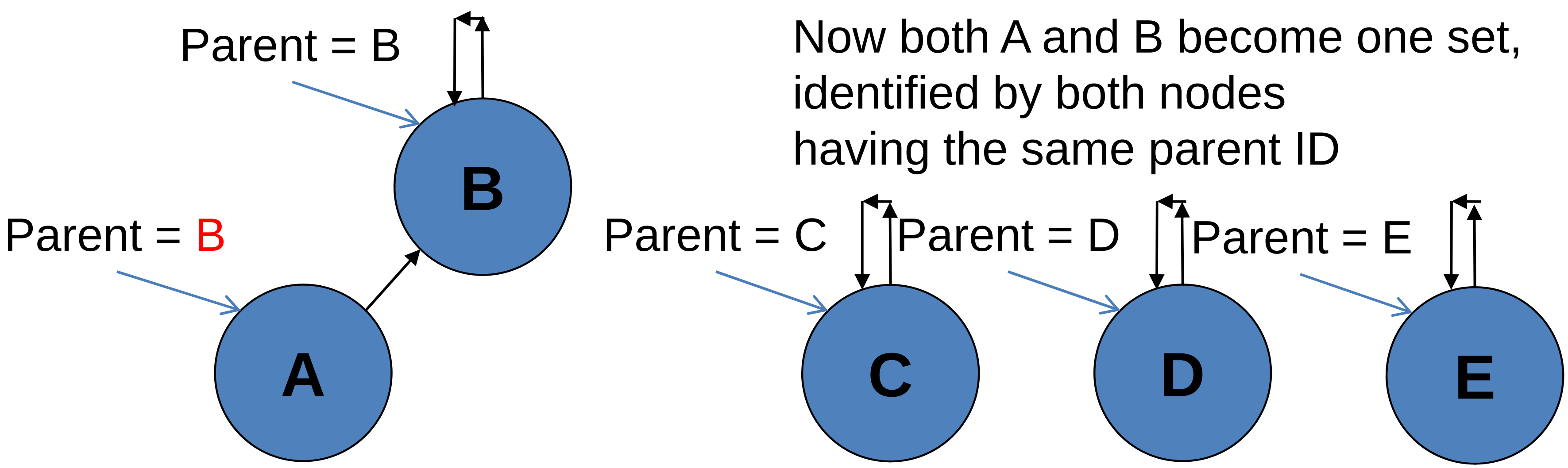
```
vector<int> pset(1000); // 1000 is just an initial number, it is user-adjustable.
void initSet(int _size) { pset.resize(_size); REP (i, 0, _size - 1) pset[i] = i; }
int findSet(int i) { return (pset[i] == i) ? i : (pset[i] = findSet(pset[i])); }
void unionSet(int i, int j) { pset[findSet(i)] = findSet(j); }
bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }

// note:
#define REP(i, a, b) \ // all codes involving REP uses this macro
  for (int i = int(a); i <= int(b); i++)
```



unionSet(A, B)

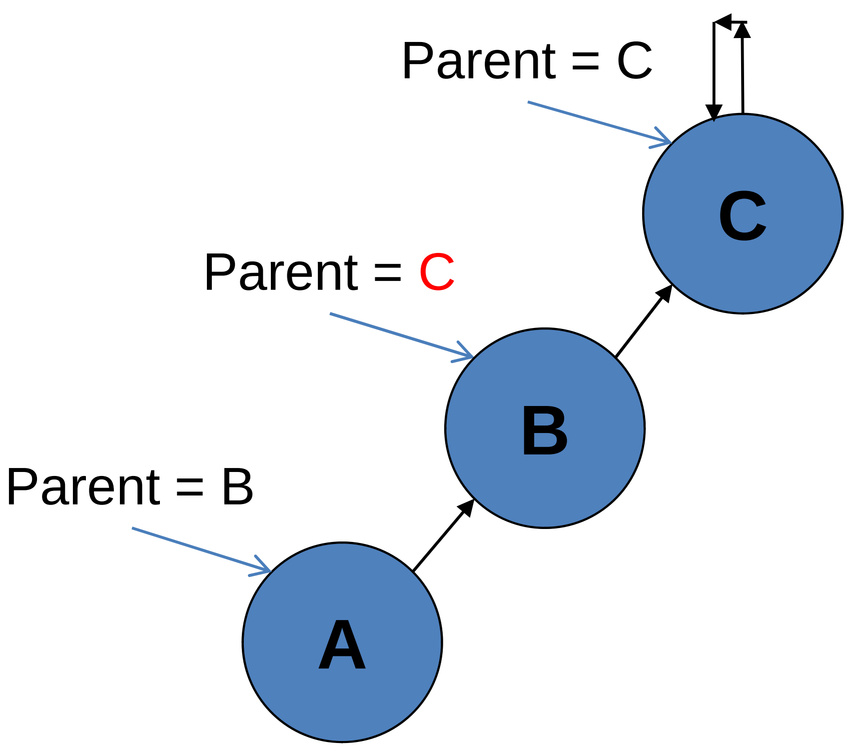
```
vector<int> pset(1000); // 1000 is just an initial number, it is user-adjustable.
void initSet(int _size) { pset.resize(_size); REP (i, 0, _size - 1) pset[i] = i; }
int findSet(int i) { return (pset[i] == i) ? i : (pset[i] = findSet(pset[i])); }
void unionSet(int i, int j) { pset[findSet(i)] = findSet(j); }
bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
```



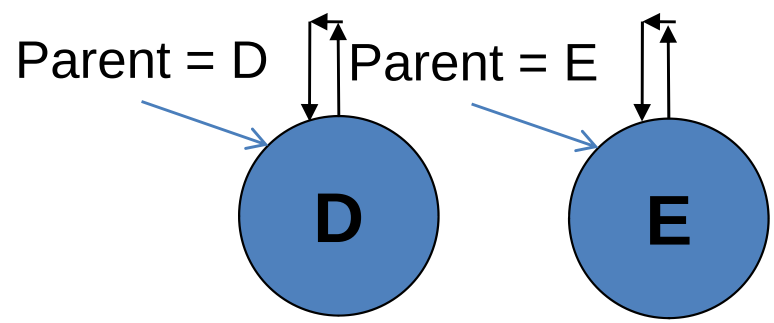
Now both A and B become one set, identified by both nodes having the same parent ID

unionSet(A, C)

```
vector<int> pset(1000); // 1000 is just an initial number, it is user-adjustable.
void initSet(int _size) { pset.resize(_size); REP (i, 0, _size - 1) pset[i] = i; }
int findSet(int i) { return (pset[i] == i) ? i : (pset[i] = findSet(pset[i])); }
void unionSet(int i, int j) { pset[findSet(i)] = findSet(j); }
bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
```

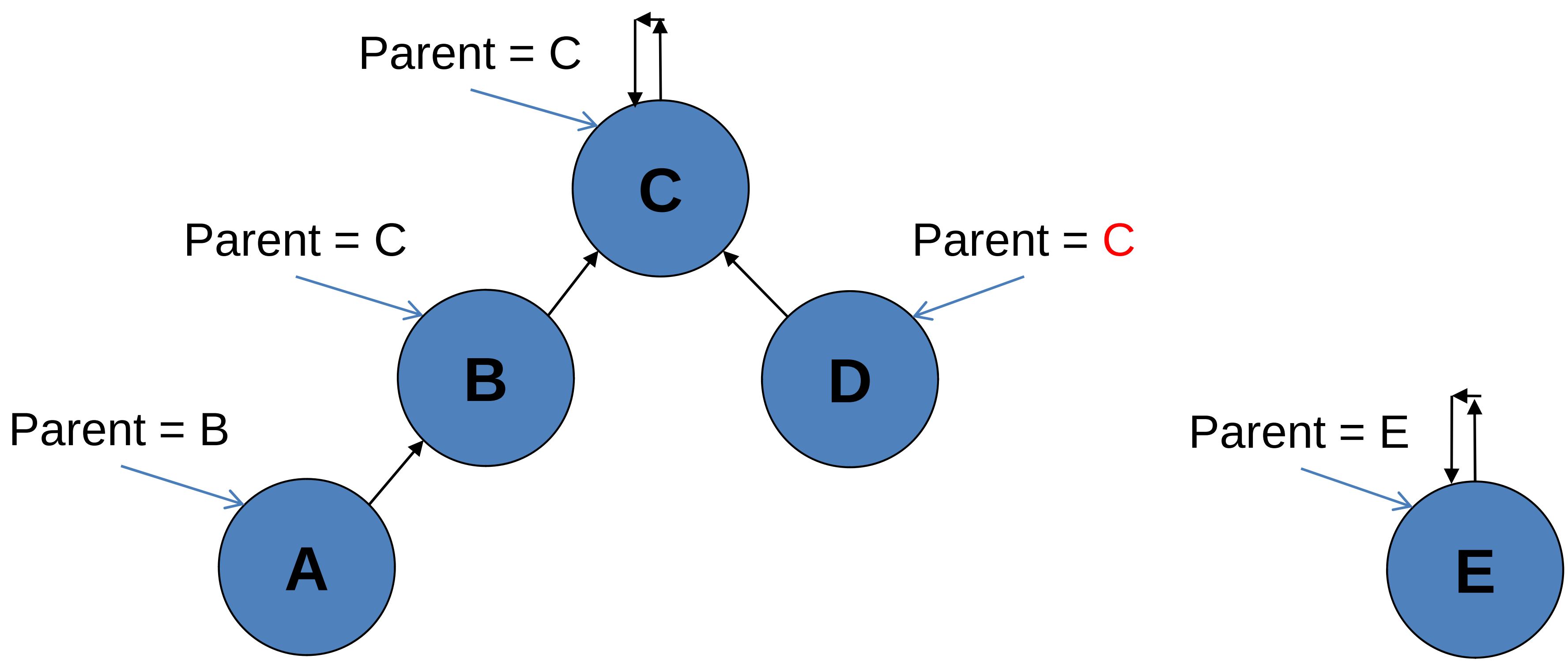


Now A, B, C become one set, identified by all three nodes having the same parent ID, directly or indirectly!



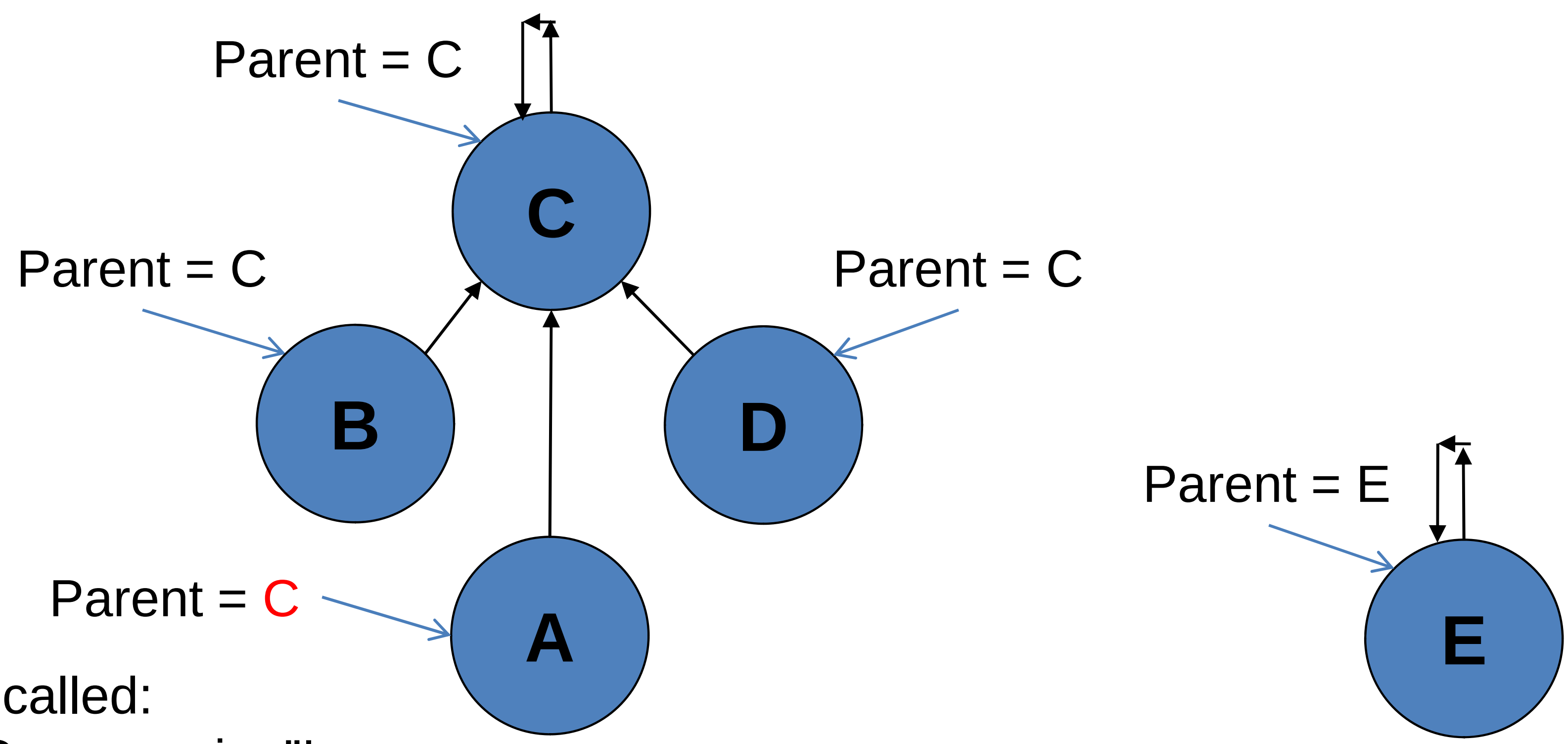
unionSet(D, B)

```
vector<int> pset(1000); // 1000 is just an initial number, it is user-adjustable.
void initSet(int _size) { pset.resize(_size); REP (i, 0, _size - 1) pset[i] = i; }
int findSet(int i) { return (pset[i] == i) ? i : (pset[i] = findSet(pset[i])); }
void unionSet(int i, int j) { pset[findSet(i)] = findSet(j); }
bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
```



findSet(A)

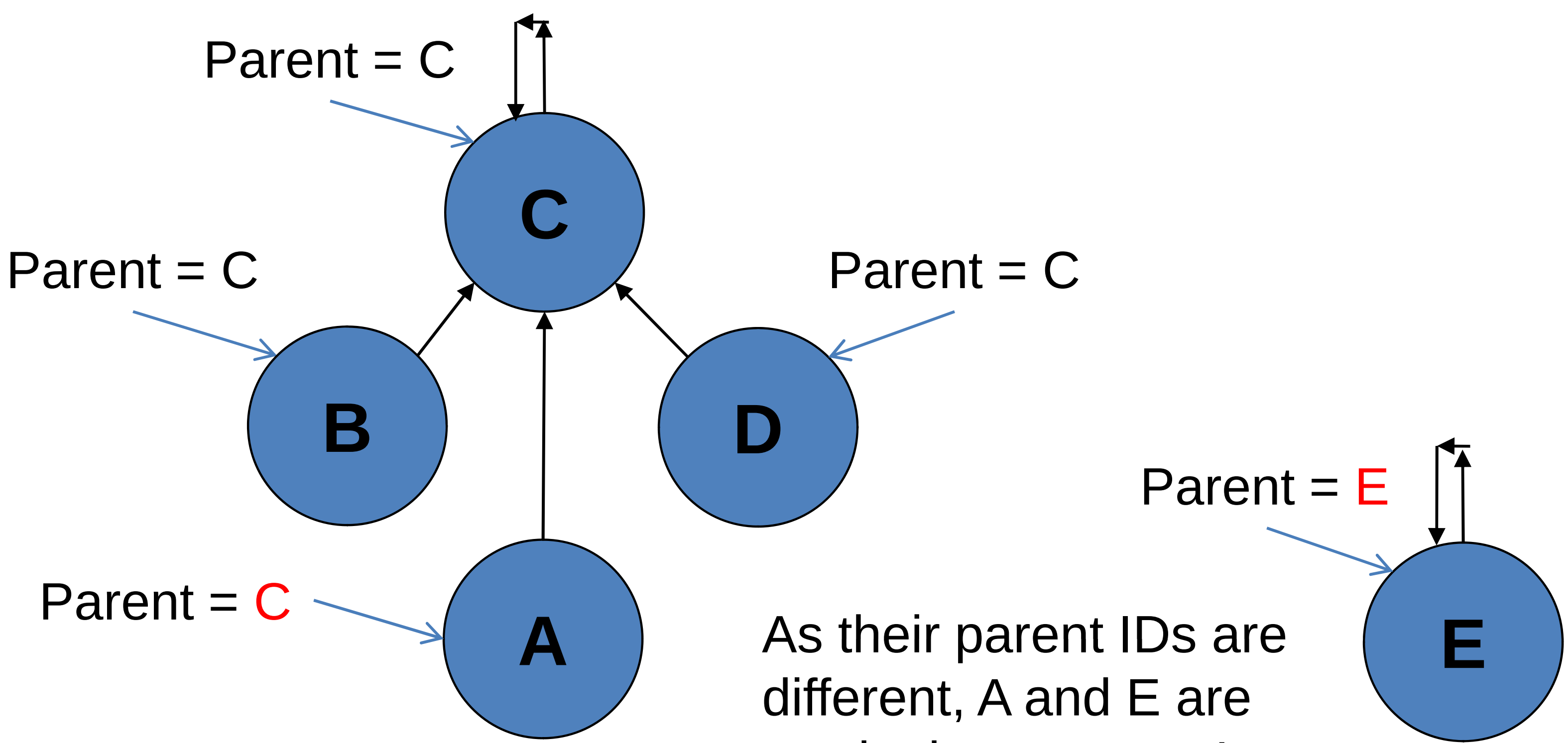
```
vector<int> pset(1000); // 1000 is just an initial number, it is user-adjustable.
void initSet(int _size) { pset.resize(_size); REP (i, 0, _size - 1) pset[i] = i; }
int findSet(int i) { return (pset[i] == i) ? i : (pset[i] = findSet(pset[i])); }
void unionSet(int i, int j) { pset[findSet(i)] = findSet(j); }
bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
```



This is called:
“Path Compression”!

isSameSet(A, E)

```
vector<int> pset(1000); // 1000 is just an initial number, it is user-adjustable.
void initSet(int _size) { pset.resize(_size); REP (i, 0, _size - 1) pset[i] = i; }
int findSet(int i) { return (pset[i] == i) ? i : (pset[i] = findSet(pset[i])); }
void unionSet(int i, int j) { pset[findSet(i)] = findSet(j); }
bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
```



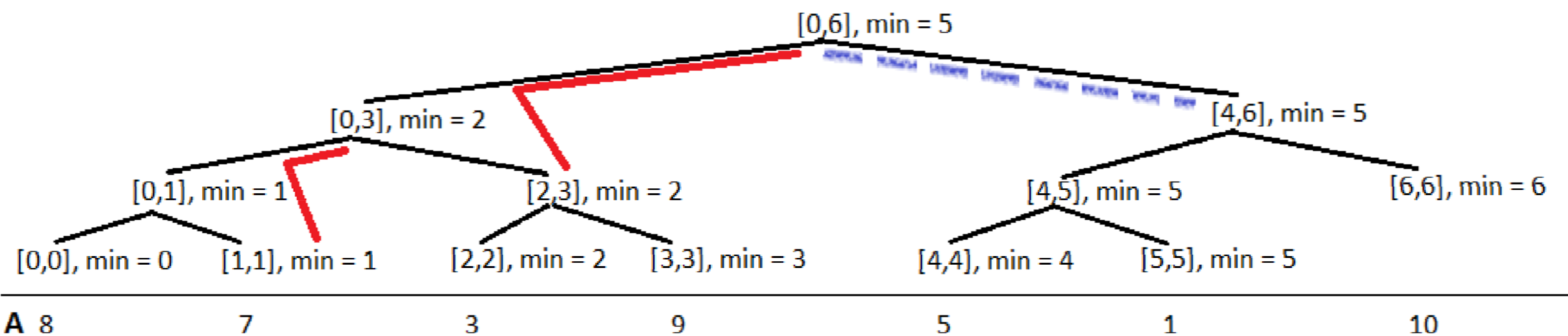
As their parent IDs are different, A and E are **not** in the same set!

Union-Find Disjoint Sets (2)

- Okay, that's the basics…
  - No *union-by-rank* or other detailed analysis…
- Further Reference:
  - **Introductions to Algorithms**, p505-509, ch21.3
  - **Algorithm Design**, p151-157, ch4.6
- No STL for this DS
  - We need to use the library shown previously
    - The code is short anyway

Segment Tree

- TBA



Fenwick Tree

- TBA

Top Coder Coding Style

SUPPLEMENTARY

Top Coder Coding Style (1)

- You may want to follow this coding style (C++)

1. Include all headers ☺
   - #include <vector>
   - #include <set>
   - #include <algorithm>
   - #include <string>
   - #include <cmath>
   - #include <queue>
   - #include <map>
   - #include <iostream>
   - #include <list>
   - #include <deque>
   - #include <cstdio>
   - #include <cstdlib>
   - using namespace std;

Want More?

Add libraries that you frequently use into this template, e.g.:

ctype.h
string.h

etc

Top Coder Coding Style (2)

2. Use shortcuts for common data types

- typedef long long ll;
- typedef long double ld;
- typedef vector<int> vi;
- typedef vector<bool> vb;
- typedef pair<int,int> ii;
- typedef vector<ii> vii;
- typedef set<int> si;

3. Simplify Repetitions/Loops!

Define your own loops style and stick with it!

- #define REP(i, a, b) for (int i = int(a); i <= int(b); i++)
- #define REPN(i, n) REP (i, 1, int(n))
- #define REPD(i, a, b) for (int i = int(a); i >= int(b); i--)
- #define TR(c, it) for (vii::iterator it = (c).begin(); it != (c).end(); it++)
- #define TR(c, it) for (typeof((c).begin()) it = (c).begin(); it != (c).end(); it++) // only for UNIX

Top Coder Coding Style (3)

4. More shortcuts
   - #define PB push_back
   - #define MP make_pair
   - #define SIZE(c) (int((c).size()))
   - #define SHOW(x) cerr << #x << " = " << x << endl;
5. STL/Libraries all the way!
   - isalpha (ctype.h)
     - inline bool isletter(char c) { return (c>='A'&&c<='Z')||(c>='a'&&c<='z'); }
   - abs (math.h)
     - inline int abs(int a) { return a >= 0 ? a : -a; }
   - pow (math.h)
     - int power(int a, int b) { int res=1; for (; b>=1; b--) res*=a; return res; }
   - Use STL data structures: vector, stack, queue, priority_queue, map, set, etc
   - Use STL algorithms: sort, lower_bound, max, min, max_element, min_element, etc

Top Coder Coding Style (4)

6. Use I/O Redirection

– int main() {
– // freopen(“input.txt", "r", stdin); // avoid re-typing the test cases!
– // freopen(“output.txt", “w", stdout); // but you may want to print to screen directly
– scanf and printf as per normal; // I prefer scanf/printf than cin/cout, C style is much easier
– }

7. Use memset effectively!

– #define INF 127 // if using memset, this is the best setting
– memset(dist, INF, sizeof(dist)); // useful to initialize shortest path distances, set INF to 127!
– memset(dp_memo, -1, sizeof(dp_memo)); // useful to initialize DP memoization table
– memset(arr, 0, sizeof(arr)); // useful to clear array of integers

8. Declare static data structure

– All input size is known, declare data structure size LARGER than needed to avoid silly bugs
– Avoid dynamic data structures that involve pointers, etc

Top Coder Coding Style (5)

- Now our coding tasks are much simpler ☺
- Typing less code = shorter coding time
  = better rank in programming contests ☺

Summary

- There are a lot of Data Structures
  - We need the most efficient one for our problem
  - Different DS suits different problem!
- Many of them have **built-in libraries**
- For some others, we have to build **our own**
  - Study these libraries! Do not rebuild them during contests!
- From Week 03 onwards and future ICPCs,
  use C++ STL and/or Java API and our built-in libraries!
  - Now, your team should be in rank 20-40 (from 60)
    (still solving ~1-2 problems out of 10, but faster)

Term Assignment Details

- TBA