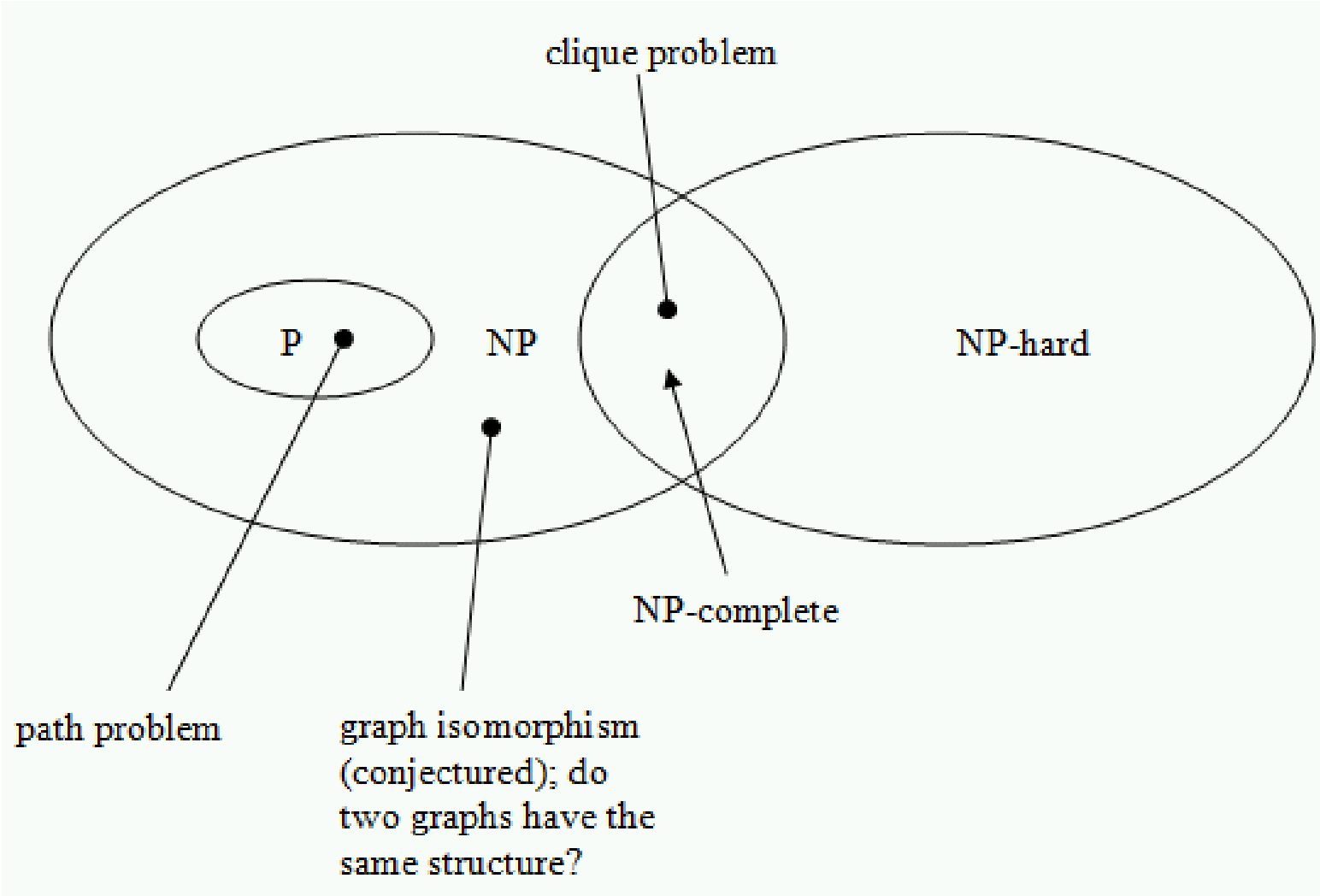
Recitation 10

Graphs/Sorts/Concurrency

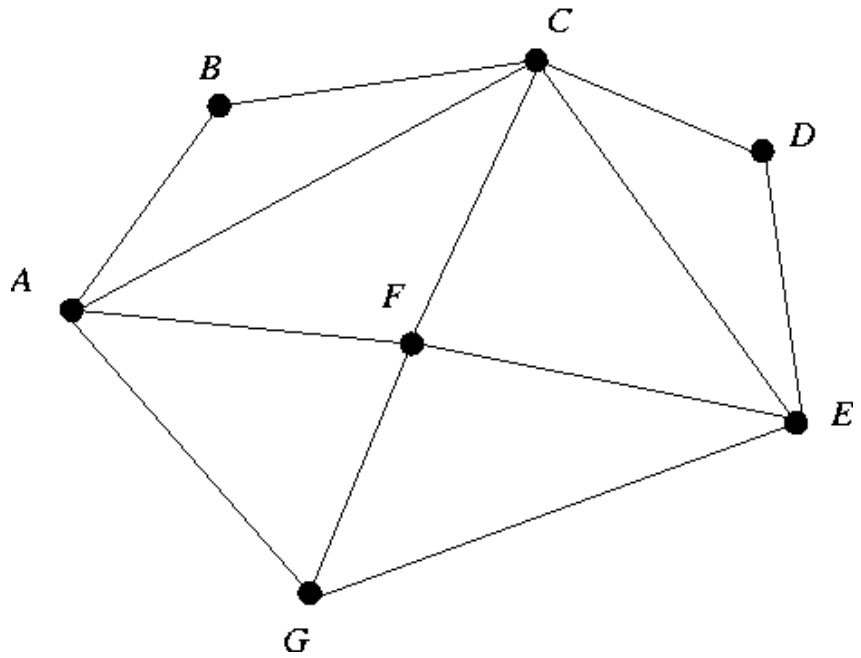
George Mappouras

11/17/2017


clique problem
P
NP
NP-hard
NP-complete
path problem
graph isomorphism (conjectured); do two graphs have the same structure?

Graphs

- Reminds of trees but now we may have multiple edges from and to different nodes
- Edges are associated with some kind of “cost” (i.e. distance)

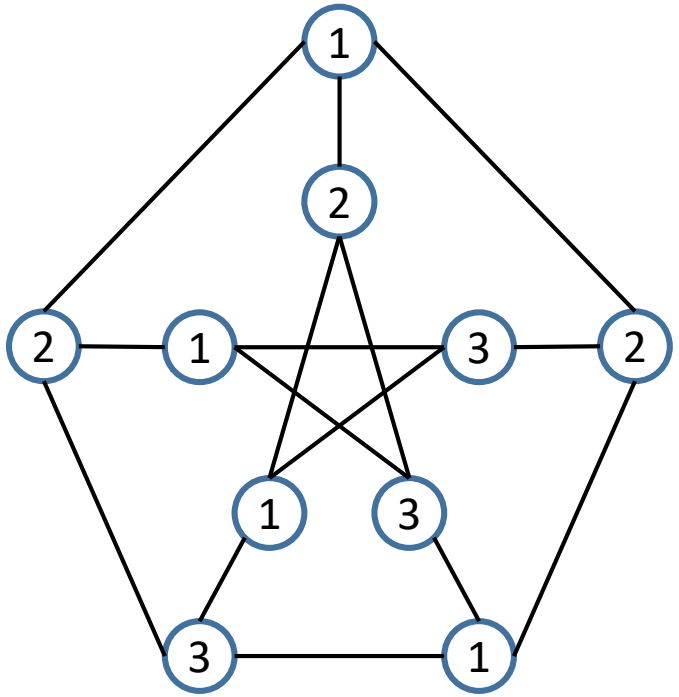


Resource allocation

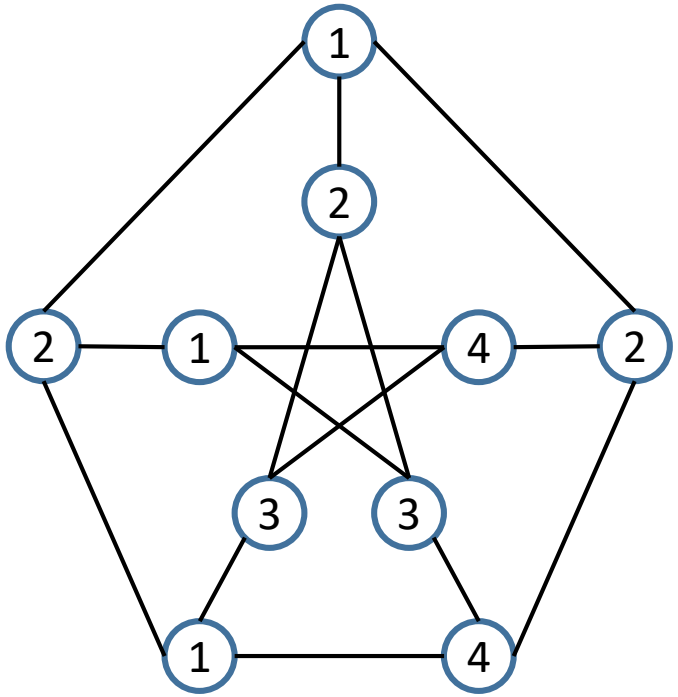
- Each edge indicates nodes that want to use a resource at the same time
- Each resource can only be used by one node each time
- Goal is to minimize the amount of resources needed
- NP-complete problem

For example: Different classes take place at different times/days. What is the minimum number of classrooms needed to schedule all ECE classes?

Or maybe not



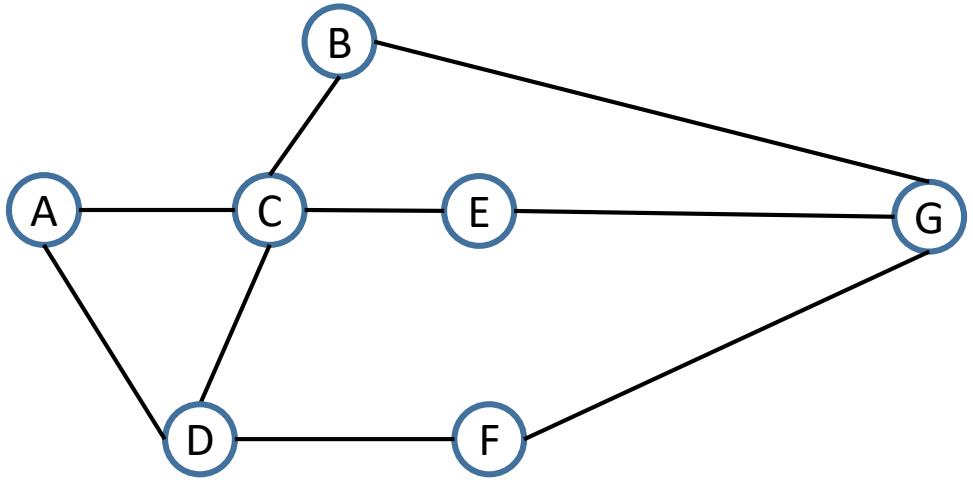
Best solution 4 colors!



Depth-First Search

How can I search for all the nodes in a graph?

How can I avoid getting in infinite loops?



Depth-First Search

Step 1: Starting node is our current node

Step 3: Current node's neighbors become visible

Step 2: If current node is not visited add new visible paths to the path stack and current node to visited

Step 3: Pop path. The last node of the path is our current node

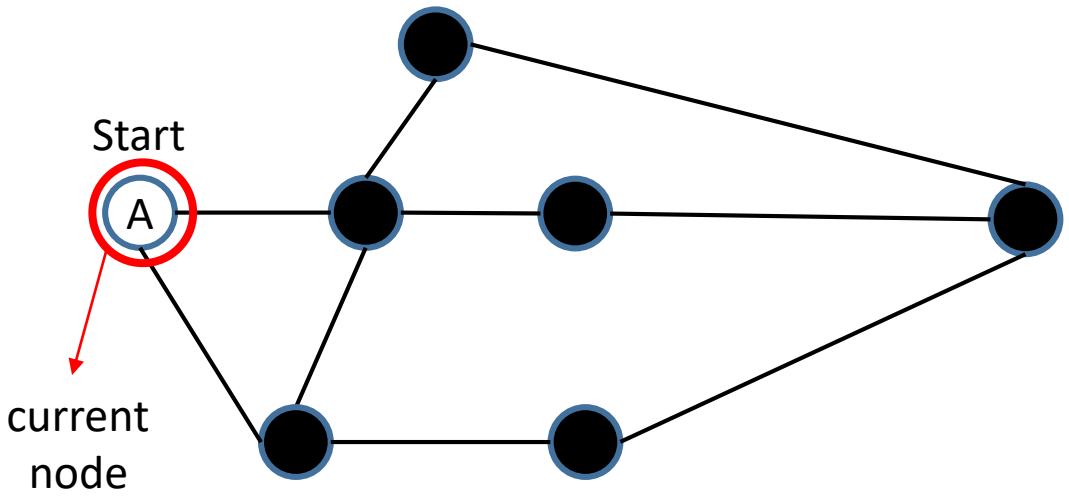
Step 4: If current node is target return path. Else go to Step 3

Depth-First Search

Visited Set

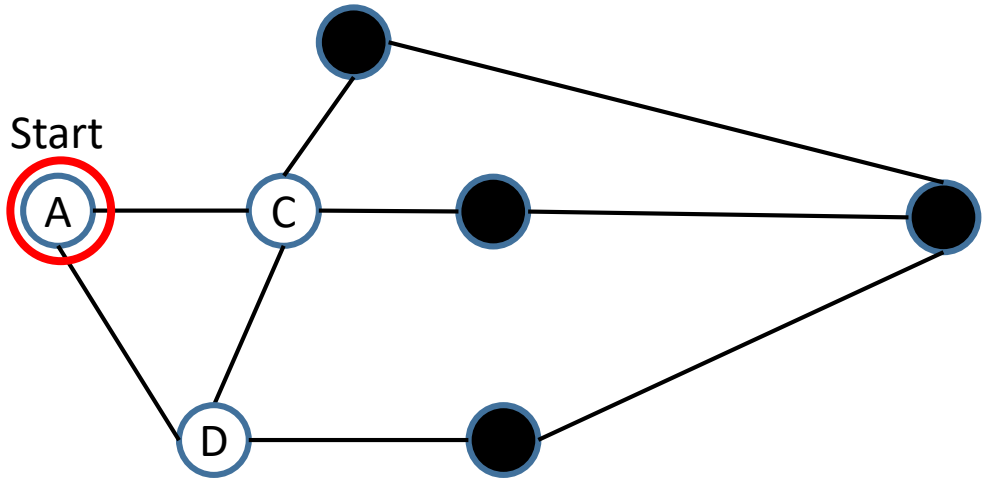
Path Stack

Looking for node G



Depth-First Search

Looking for node G



Visited Set

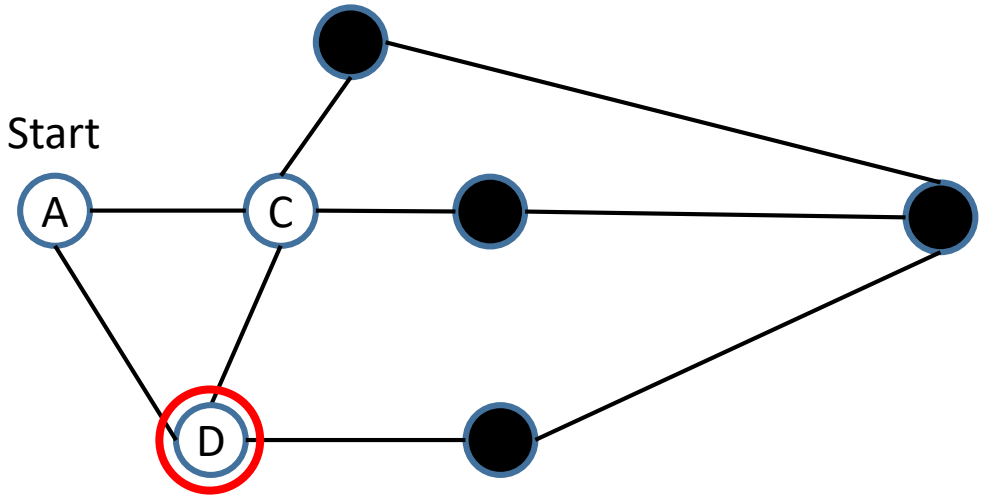
A

Path Stack

AD
AC

Depth-First Search

Looking for node G



Visited Set

A D

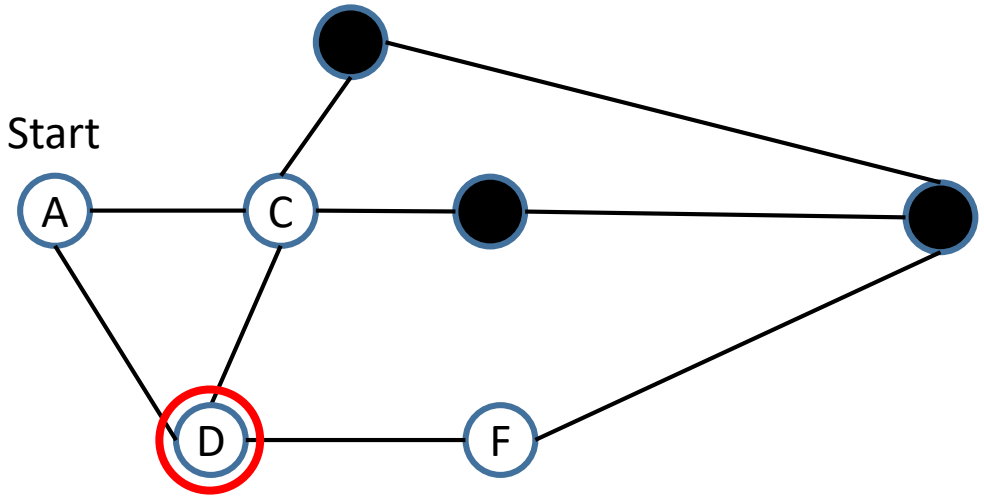
Path Stack

AC

AD

Depth-First Search

Looking for node G



Visited Set

A D

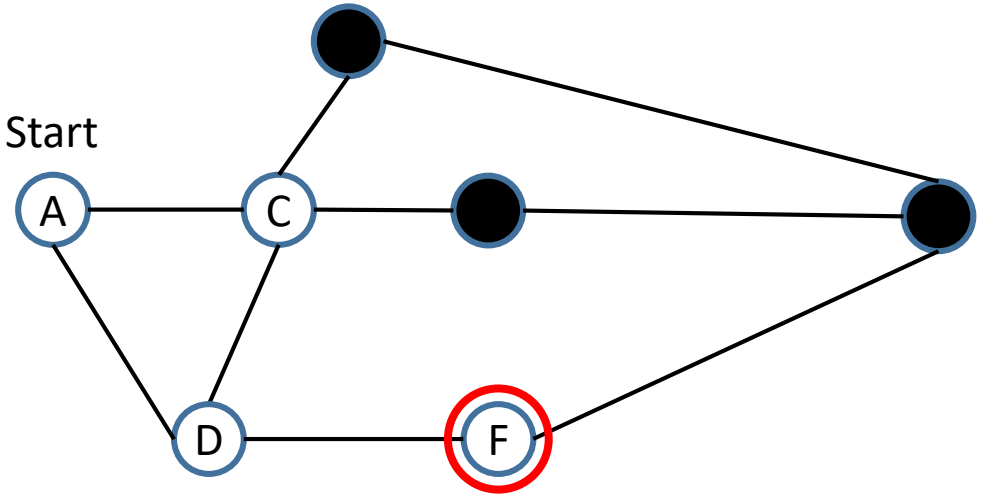
Path Stack

ADF
ADC
ADA
AC

AD

Depth-First Search

Looking for node G



Visited Set

A D F

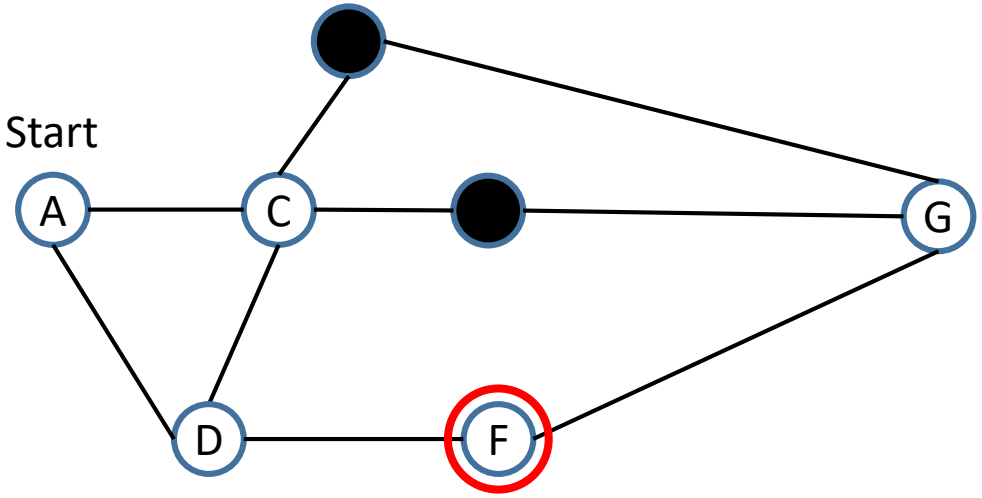
Path Stack

ADC
ADA
AC

ADF

Depth-First Search

Looking for node G



Visited Set

A D F

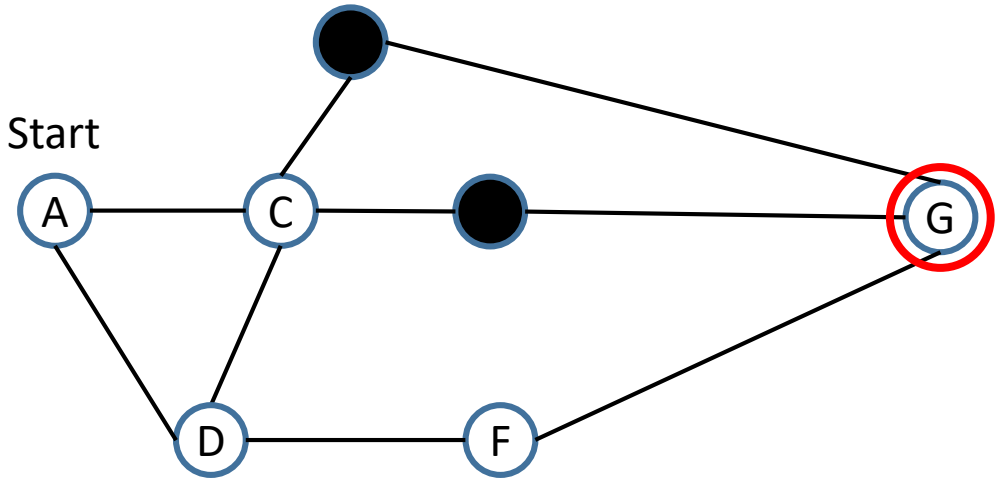
Path Stack

ADFG
ADFD
ADC
ADA
AC

ADF

Depth-First Search

Looking for node G



Visited Set

A D F G

Path Stack

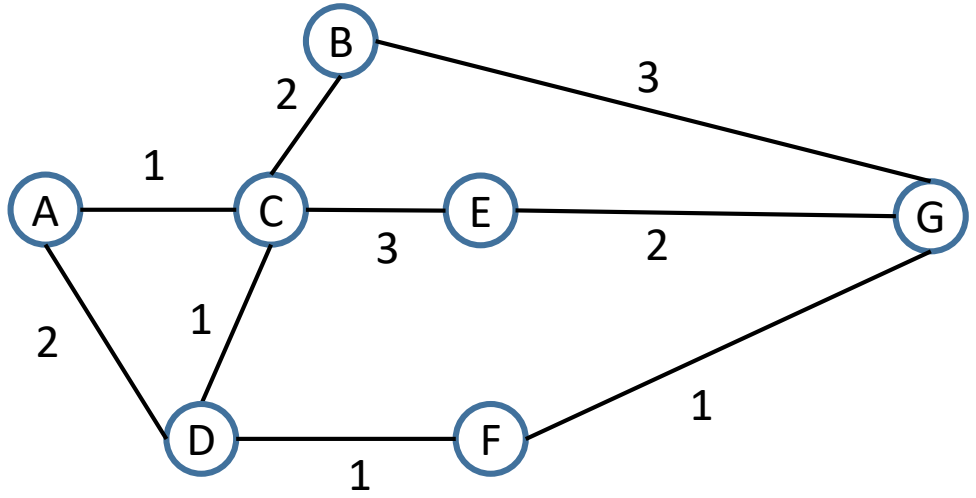
ADFD
ADC
ADA
AC

ADFG

Path planning

1) Find the “shortest” path starting from a specific node to all other nodes

2) What is the shortest path to visit all nodes, starting from any node and with no cycles (TSP)?



Solving (1)

Dijkstra's algorithm:

Step 0: Starting node has 0 cost, all others inf

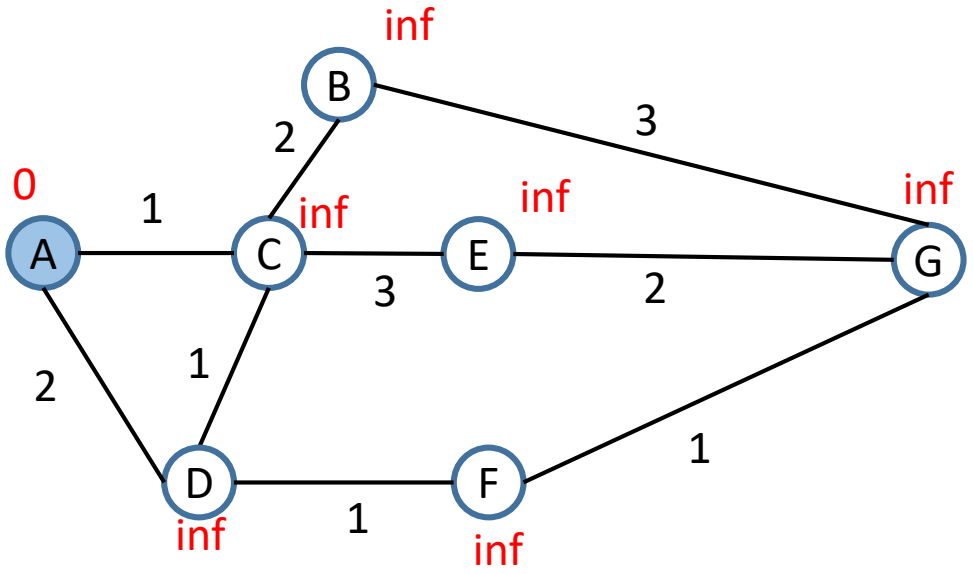
Step 1: Next node is starting point

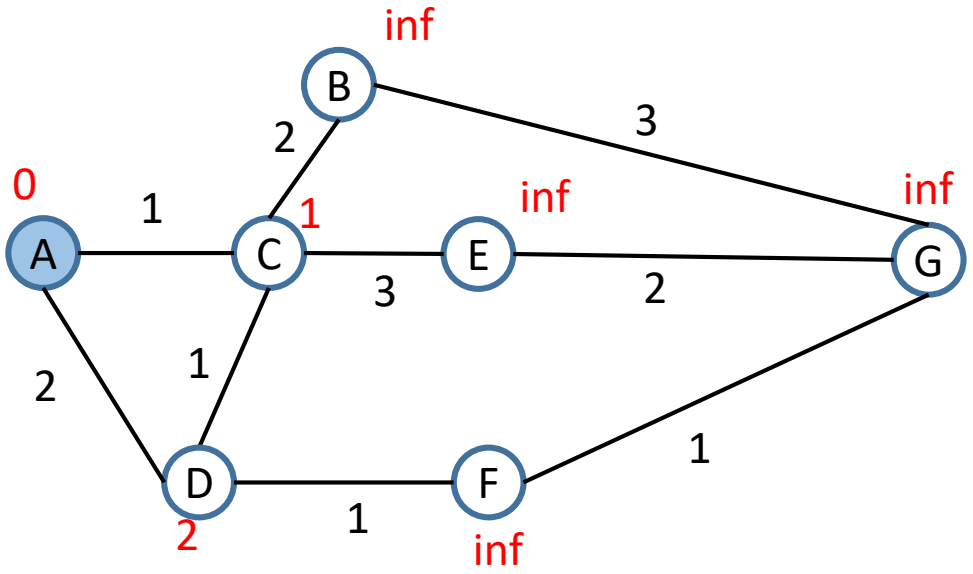
Step 2: Visit next node

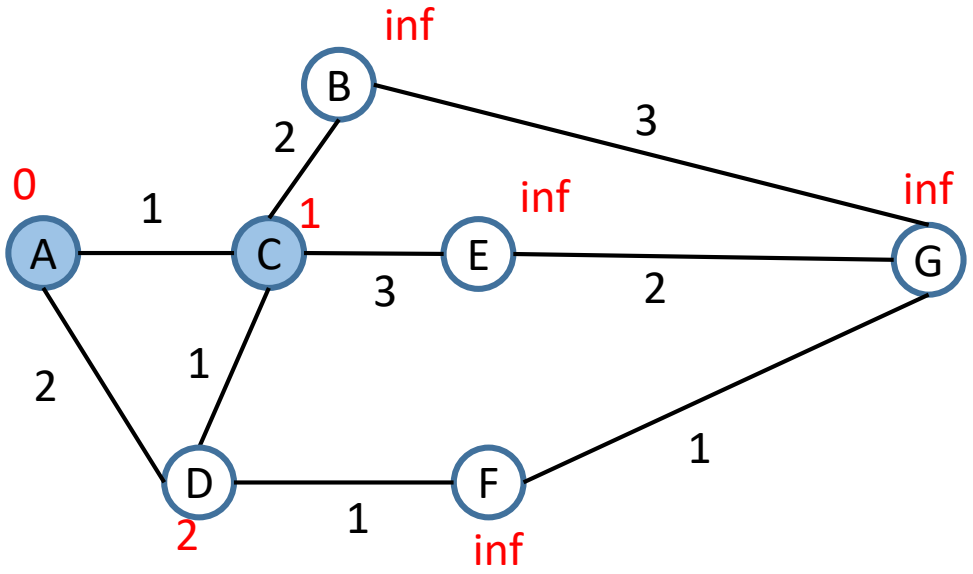
Step 3: Check all the neighbors of the visited node and calculate the cost to reach each node from the current node (current nodes cost + edge cost). If the new cost of a neighbor is less than their current cost update it.

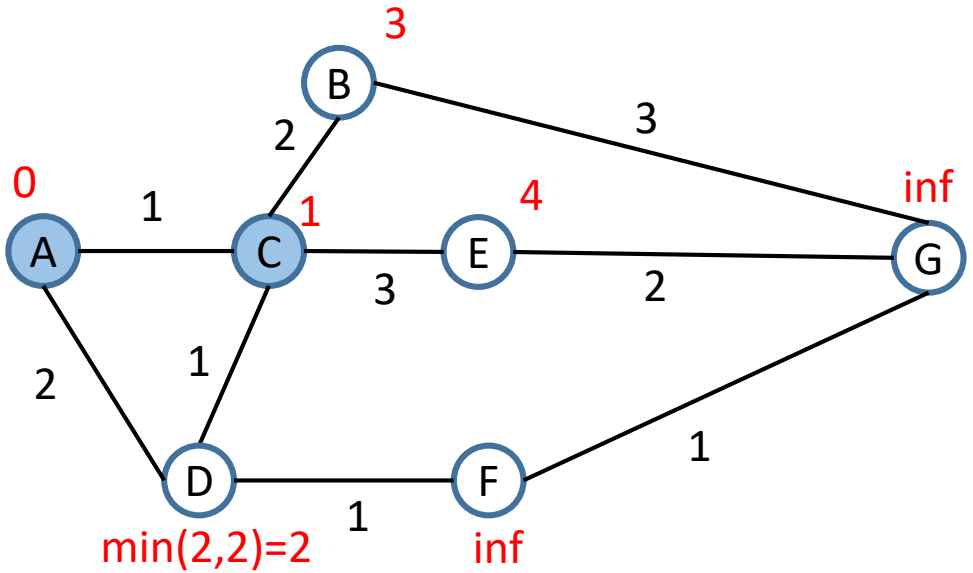
Step 4: Next node is the node with smallest cost that is not visited

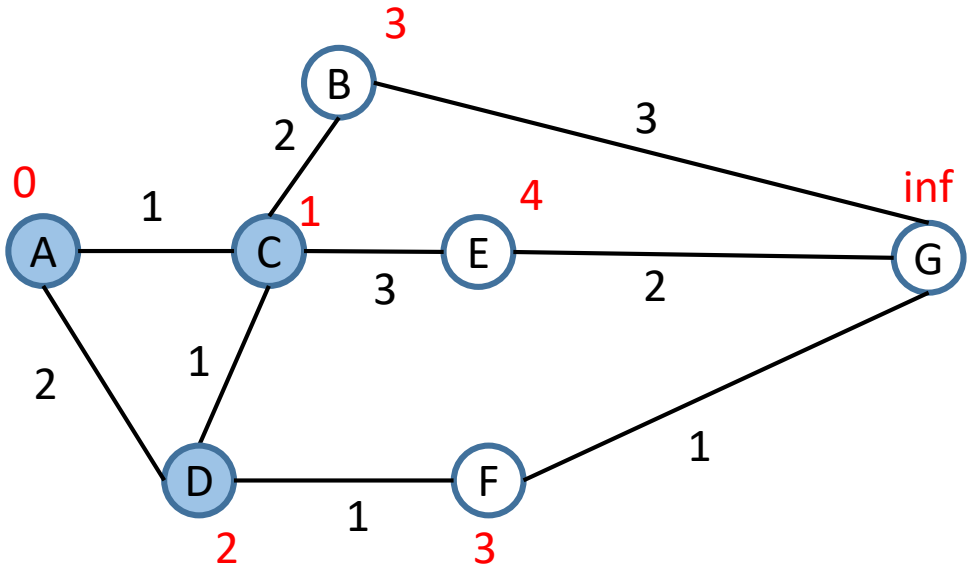
Step 5: Go to Step 2

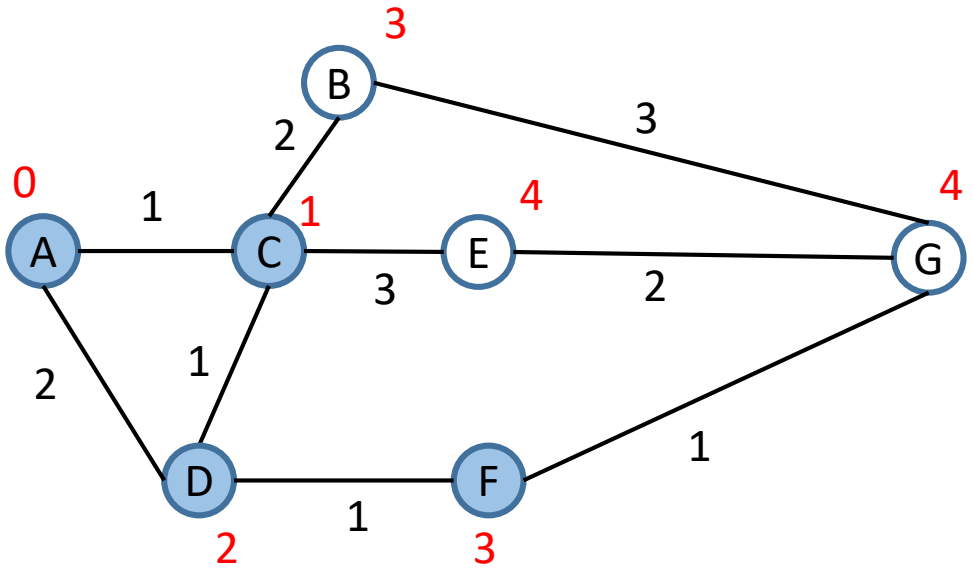

inf
B
3
2
0
1
inf
inf
inf
A
C
E
G
3
2
1
2
1
D
F
1
inf
inf

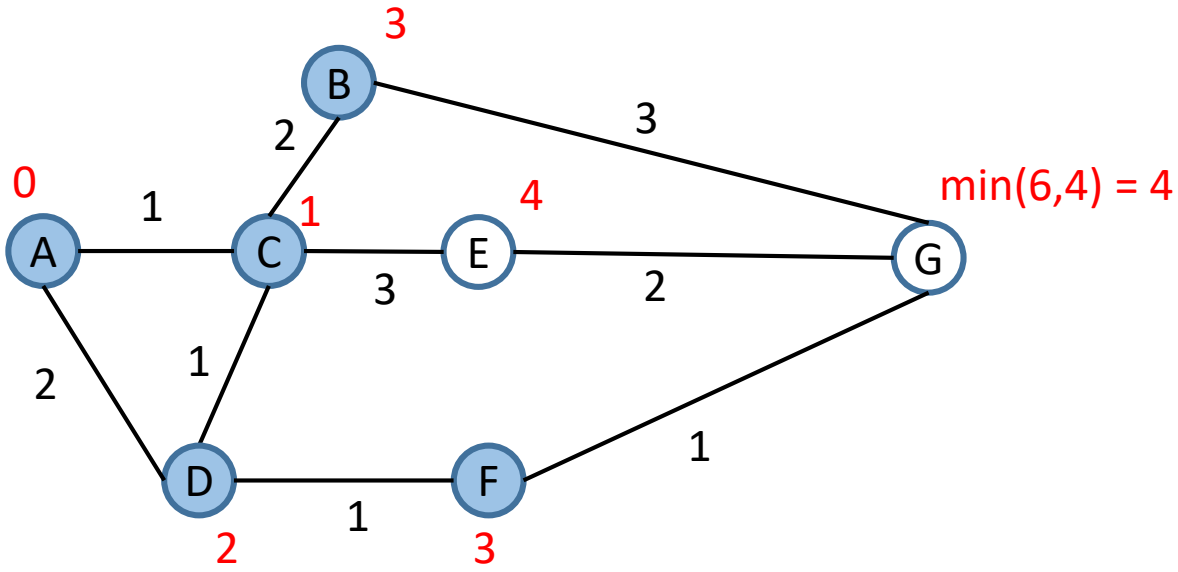

inf
B
3
2
0
1
1
inf
inf
A
C
E
G
3
2
1
2
D
F
1
1
2
inf

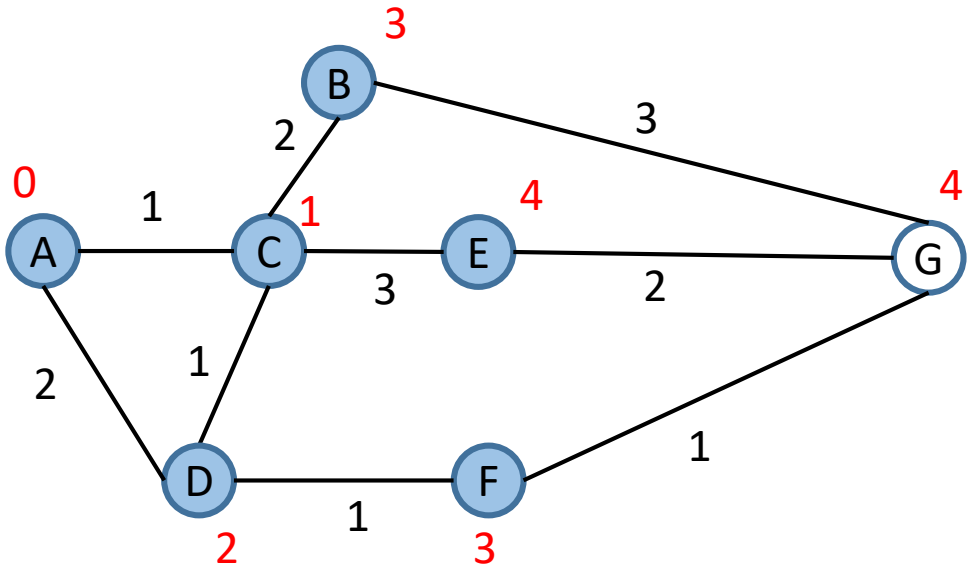

inf
B
3
2
0
1
1
inf
inf
A
C
E
G
3
2
1
2
1
D
F
1
2
1
inf

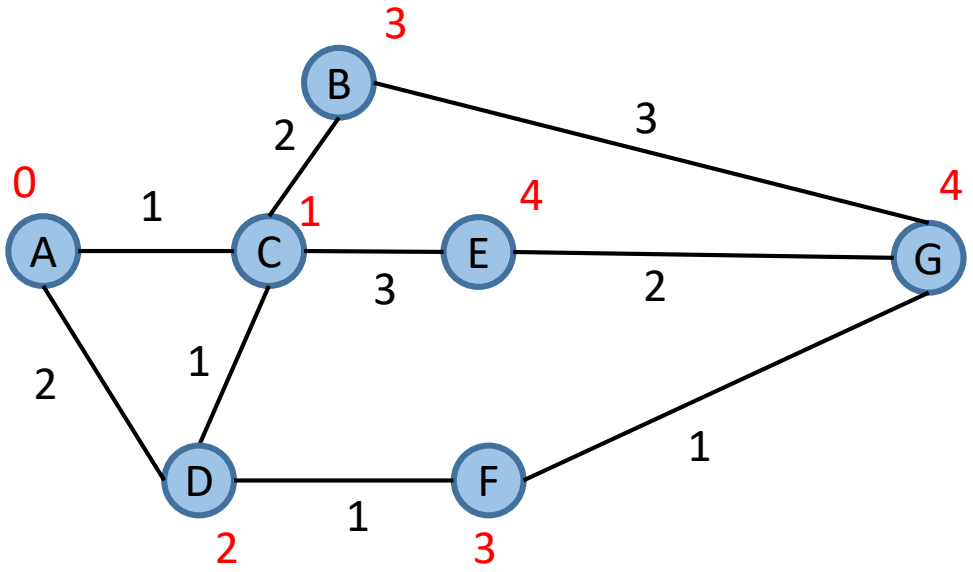

3
B
3
2
0
1
1
4
inf
A
C
E
G
3
2
2
1
D
F
1
1
min(2,2)=2
inf

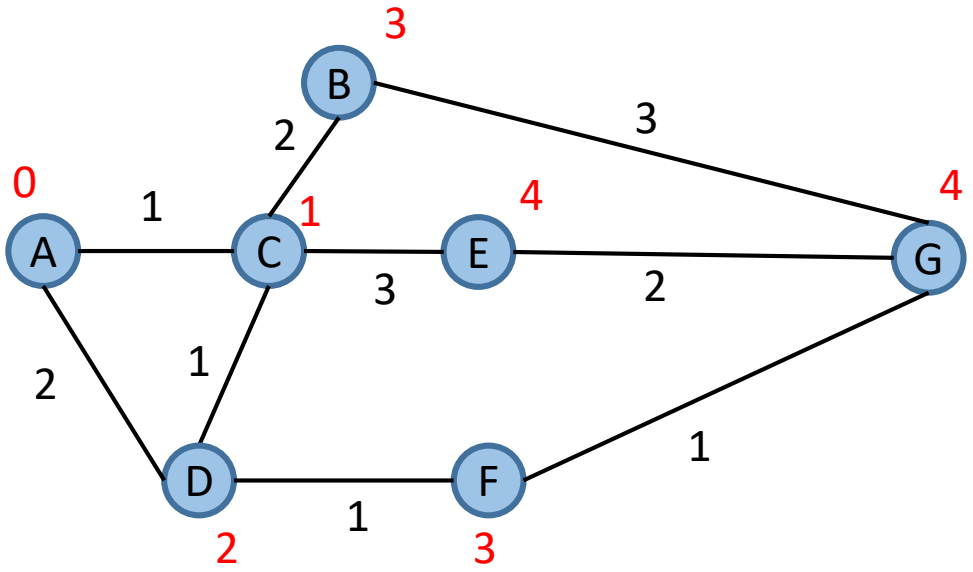

3
B
3
2
0
1
1
4
inf
A
C
E
G
3
2
2
1
D
F
1
1
2
3

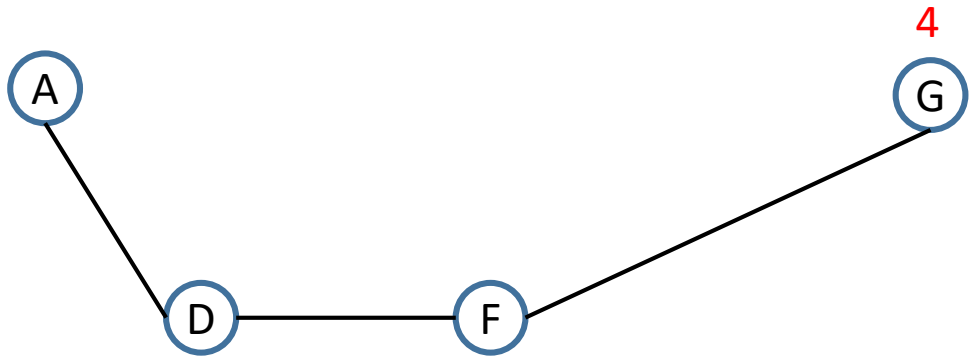

3
B
3
2
0
1
1
4
4
A
C
E
G
3
2
2
1
1
D
F
2
1
3


3
B
3
2
0
1
1
4
min(6,4) = 4
A
C
E
G
3
2
1
2
1
D
F
1
2
3


3
B
3
2
0
1
1
4
4
A
C
E
G
3
2
2
1
1
D
F
2
1
3


3
B
3
2
0
1
1
4
4
A
C
E
G
3
2
2
1
1
D
F
2
1
3


3
B
3
2
0
1
1
4
4
A
C
E
G
3
2
2
1
1
D
F
2
1
3


4
A
G
D
F

Dijkstra’s run time?

Let us assume V nodes and E edges

How many times do I need to visit each node in the worst case?

O(V^2)

Minimum Spanning Tree

Given a graph, find a minimum tree that includes all nodes. Meaning the total costs of the edges of the tree is minimum

Adding some other edge and removing an existing one will result to an increased cost or not including all nodes

Solution: Prim’s Algorithm

Prim’s Algorithm

Step 1: Start from any node

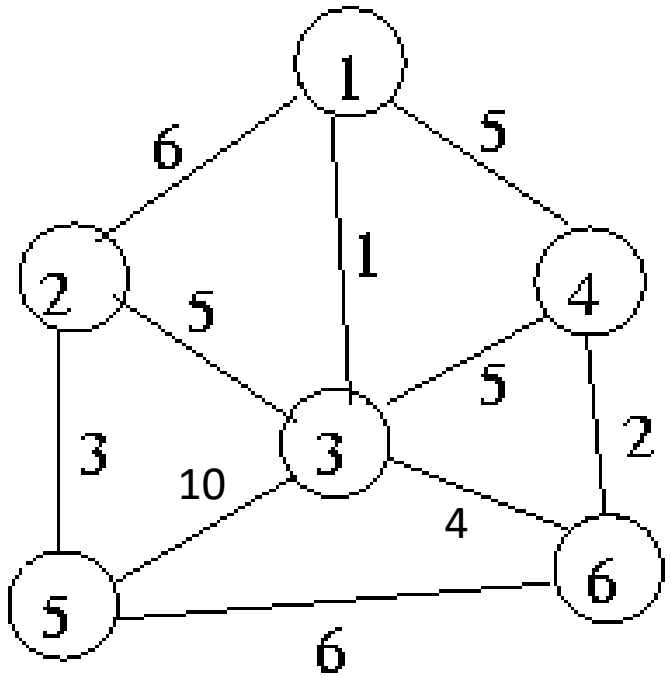
Step 2: Add node to visited list

Step 3: Find the edge with the minimum cost that can be visited with one step from any of our visited nodes and leads to an unvisited node

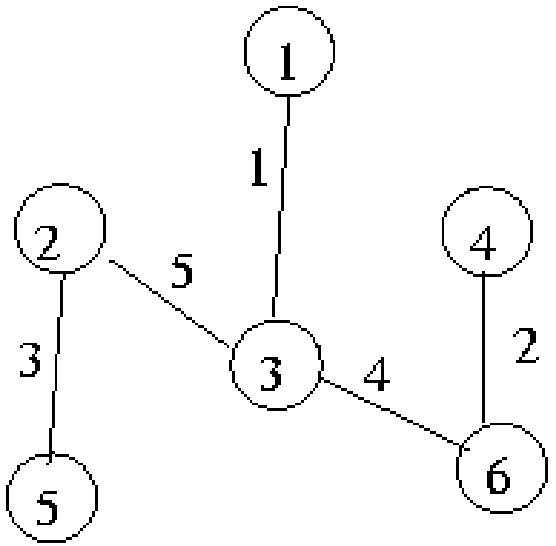
Step 4: Visit the node of that edge and add it to your list

Step 5: Go to Step 3

Prim's Example



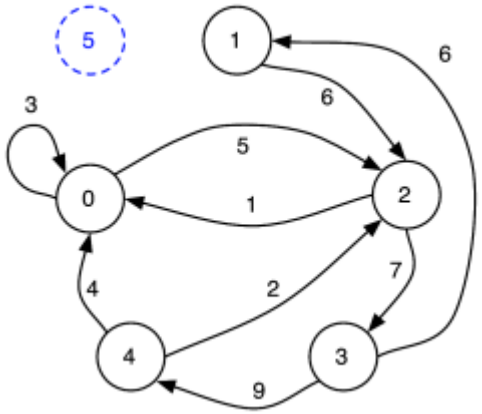
Solution to Prim's Example



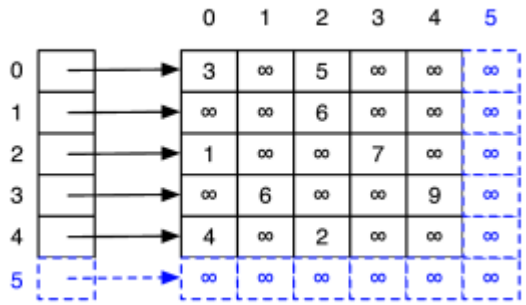
How to implemental Graphs?

We have 3 options

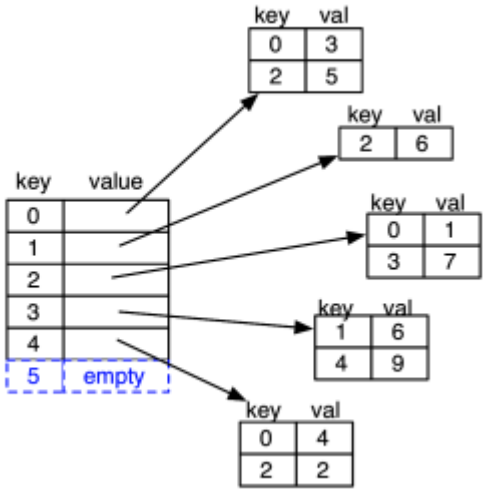
Nodes



2D Tables



Lists



Other Problems

- Traveling Salesperson (TSP)

The shortest path to traverse all the nodes only once

- Clique

number of nodes that are fully connected

- Max-flow Min-cut

The minimum number of edges that if we remove the graph is no longer connected

```
int * dijkstra(int graph[V][V], int start, int * dist)
{
    bool visited[V];
    for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, visited[i] = false;

    dist[start] = 0;

    for (int count = 0; count < V-1; count++)
    {
     int u = minDistance(dist, &visited);
     visited[u] = true;
     for (int v = 0; v < V; v++)
           if (!visited[v] && graph[u][v] && dist[u] != INT_MAX
                           && dist[u]+graph[u][v] < dist[v])
               dist[v] = dist[u] + graph[u][v];
    }
    return dist;
}
```

```
int minDistance(int * dist, bool * visited)
{
  int min = INT_MAX, min_index;

  for (int v = 0; v < V; v++)
    if (visited[v] == false && dist[v] <= min)
        min = dist[v], min_index = v;

  return min_index;
}
```

```
void primMST(int graph[V][V], int * parent)
{
    int key[V];
    bool visited[V];

    for (int i = 0; i < V; i++)
      key[i] = INT_MAX, visited[i] = false;

    key[0] = 0;
    parent[0] = -1; // It's the root of the MST

    for (int count = 0; count < V-1; count++)
    {
      int u = minKey(key, visited);
      visited[u] = true;

      for (int v = 0; v < V; v++)
        if (graph[u][v] && visited[v] == false && graph[u][v] <  key[v])
          parent[v]  = u, key[v] = graph[u][v];
    }
 }
```

Sorting

Insertion Sort

Step1: Select number_k

Step2: i=k if number_i-1 > number_i => i = i -1 and repeat Step2

else go to Step 3

Step3: Swap number_k with number_i

Step3: k = k+1 –> Go to Step 1

Sorting

Insertion Sort – example

initial array: 5, 10, 2, 3, 13, 6,

k=1, i=1: 5, 10, 2, 3, 13, 6 => stop i=k

k=2,i=1: 10, 5, 2, 3, 13, 6 k=2,i=2: 5, 10, 2, 3, 13, 6 => stop i=k

k=3,i=1: 2, 5, 10, 3, 13, 6 => stop 2<5

k=4, i=1: 3, 2, 5, 10, 13, 6 k=4, i=2: 2, 3, 5, 10, 13, 6 => stop 3<5

…

Sorting

Insertion Sort

What is the big-O?

Sorting

Insertion Sort

What is the big-O?

O(N^2)

Heap Sort

Step 1: For each element of the array insert it in a heap

Step 2: After you insert all elements in the heap, remove the root of the tree and place it to the array

Step 3: The array is now sorted

Heap Sort

What is the big-O now?

We perform heap insertions and heap root removes…

Heap Sort

What is the big-O now?

We perform heap insertions and heap root removes… so:

heap insertion = log(N)

root remove = log(N)

Heap Sort

What is the big-O now?

We perform heap insertions and heap root removes… so:

N heap insertions and N root removes

Heap Sort

What is the big-O now?

We perform heap insertions and heap root removes… so:

N*log(N) + N*log(N) = 2Nlog(n) => O(N*log(N))!

Sorting with Divide&Conquer

Merge Sort:

Divide the problem to sorting of smaller arrays. Merge the results in to a final bigger array

Quick Sort:

Similar to merge. We want to embed some intuition of how to divide the problem to smaller problems. We use a “pivot” to create the smaller arrays.

Concurrency

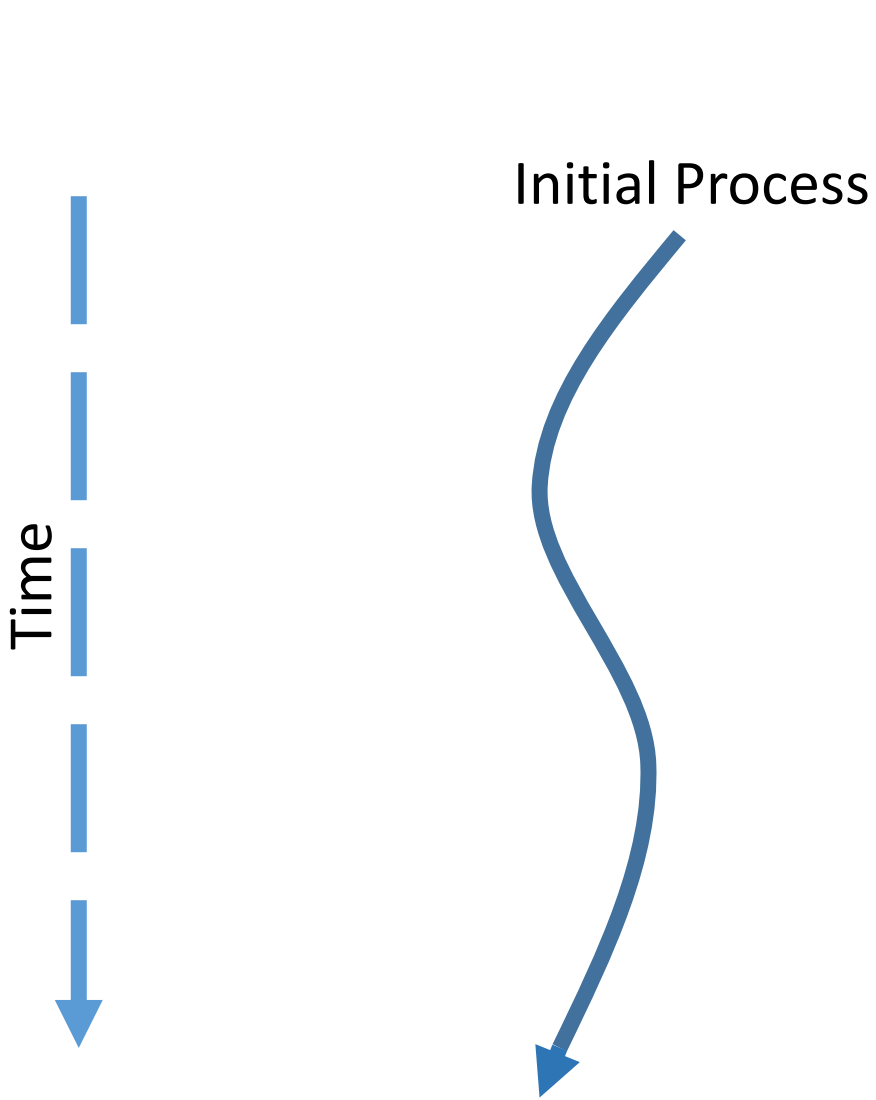
- Why do we have multi-core CPUs?

  Executing multiple programs in parallel

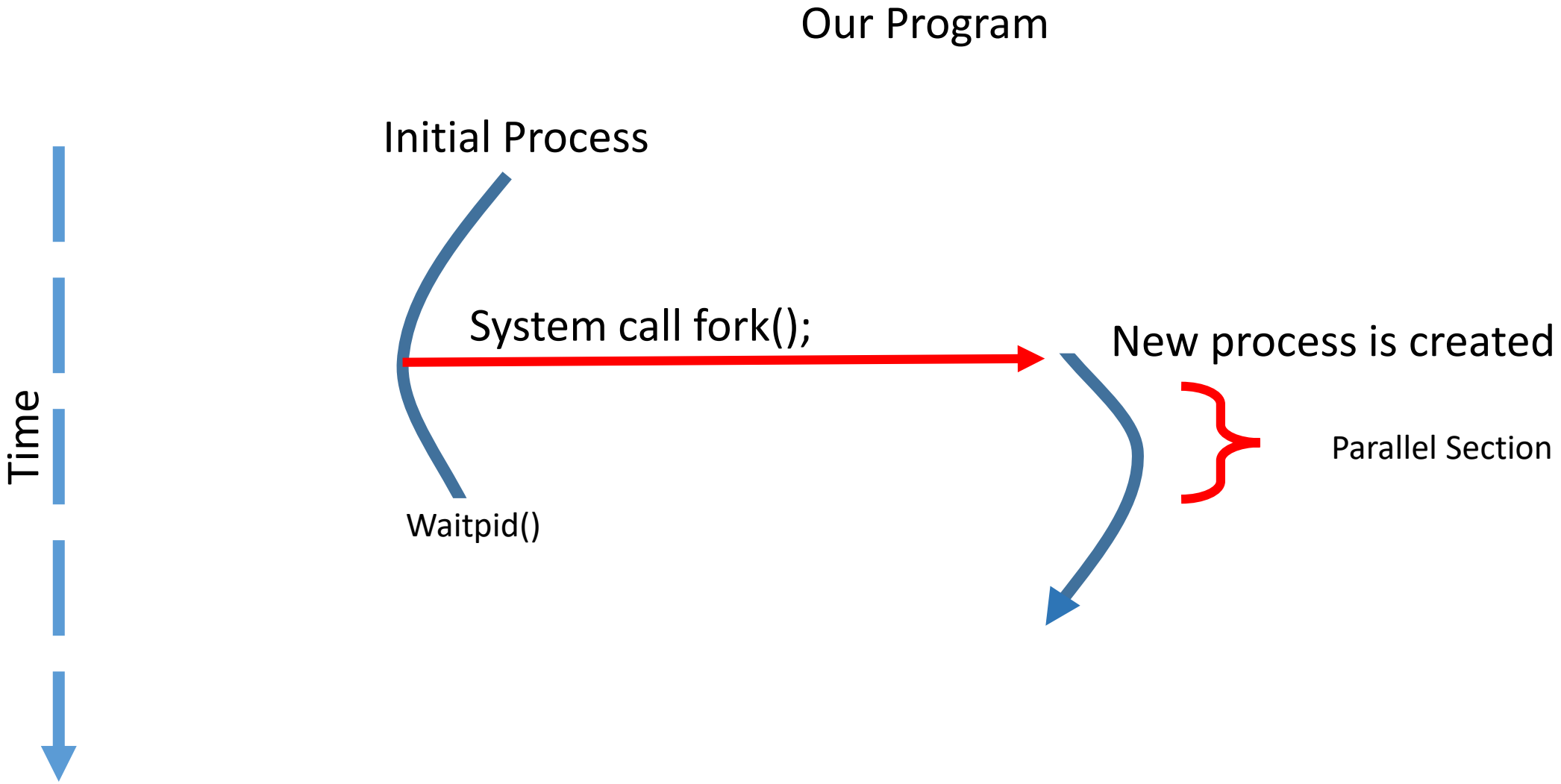
- What about a single program? Can parts of it be executed in parallel from different cores

  Of course they can! But we need to write a parallel program.

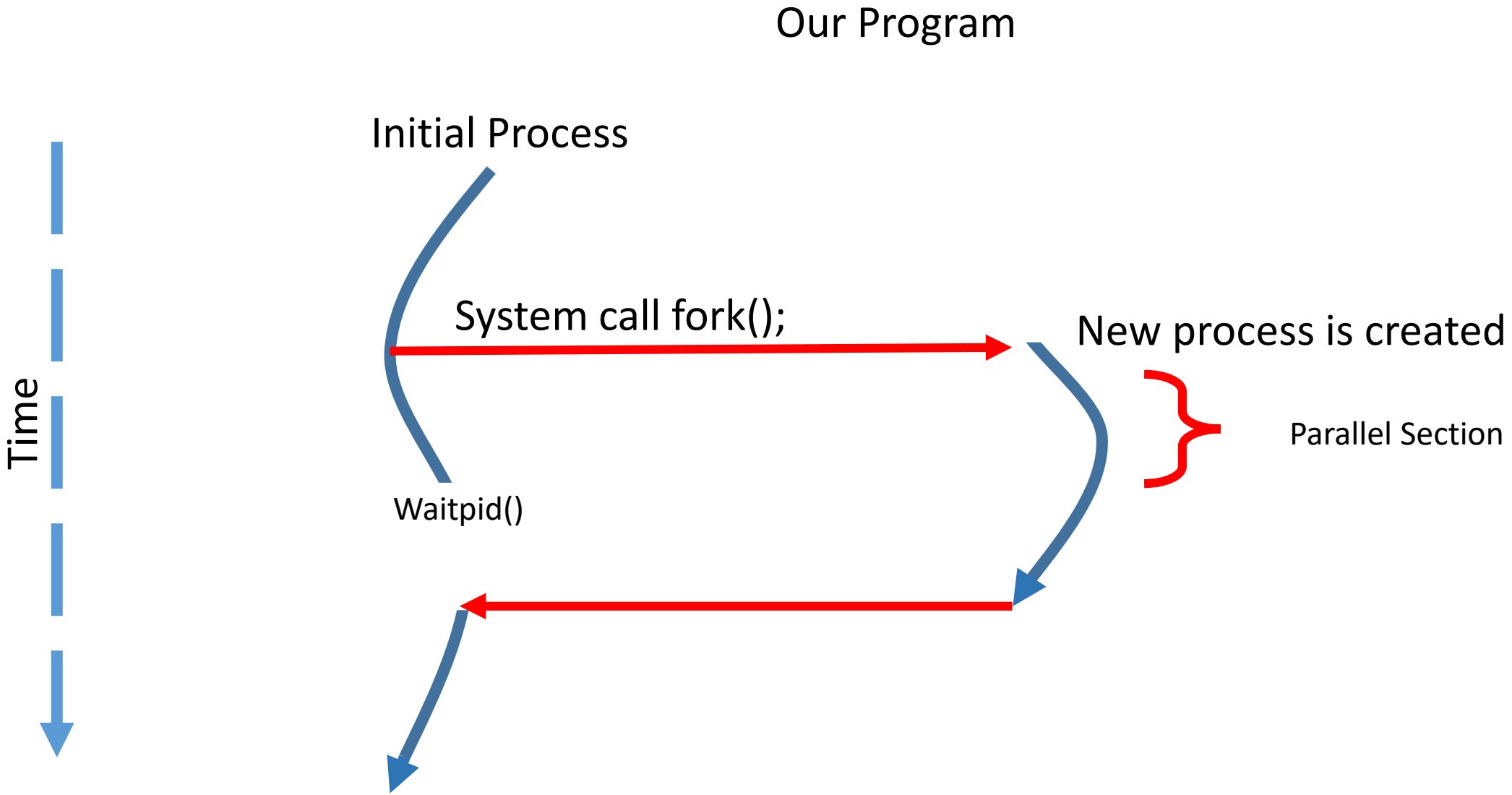
Using fork()



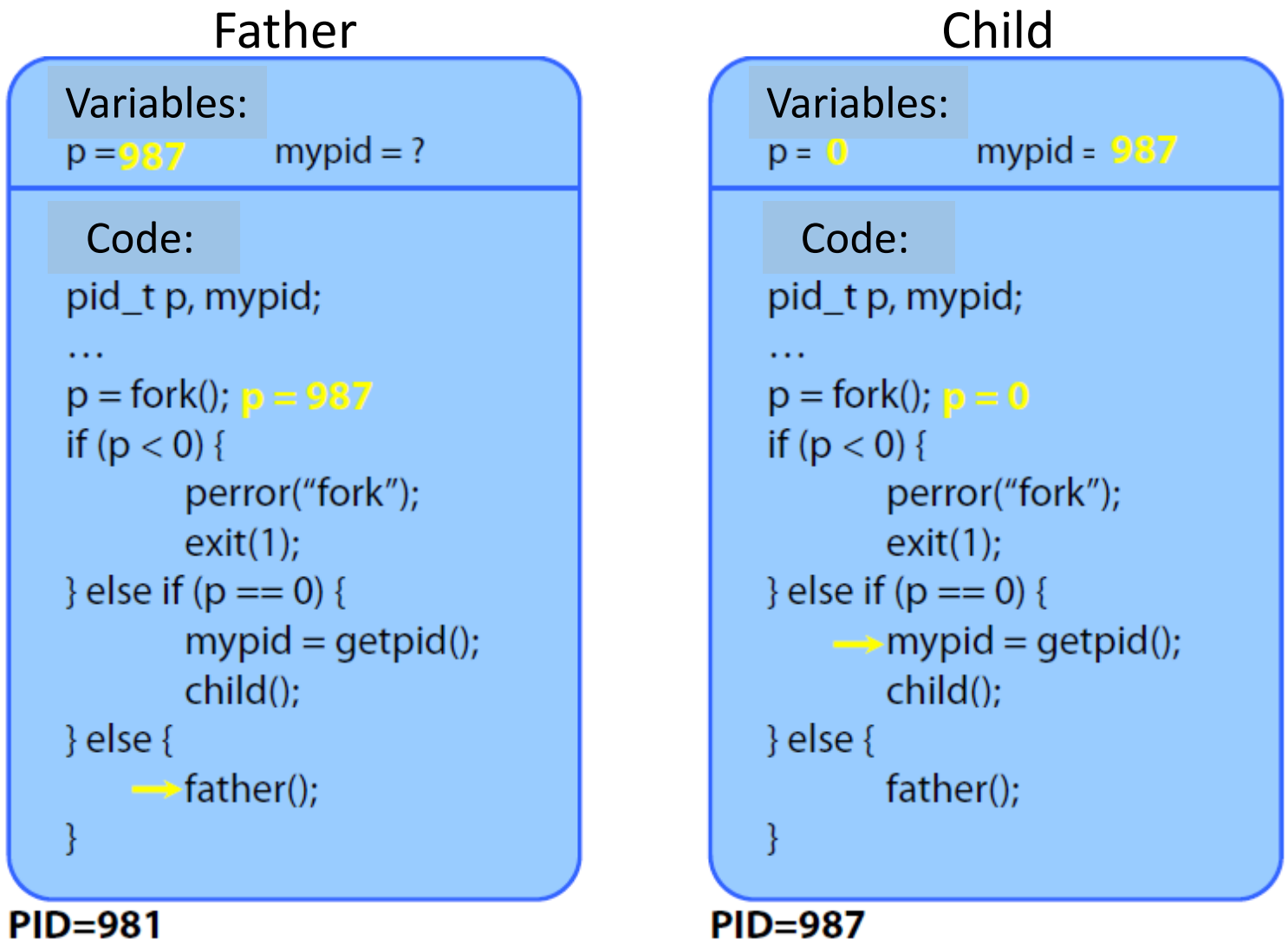
Using fork()


Our Program
Initial Process
System call fork();
New process is created
Parallel Section
Waitpid()
Time

Using fork()

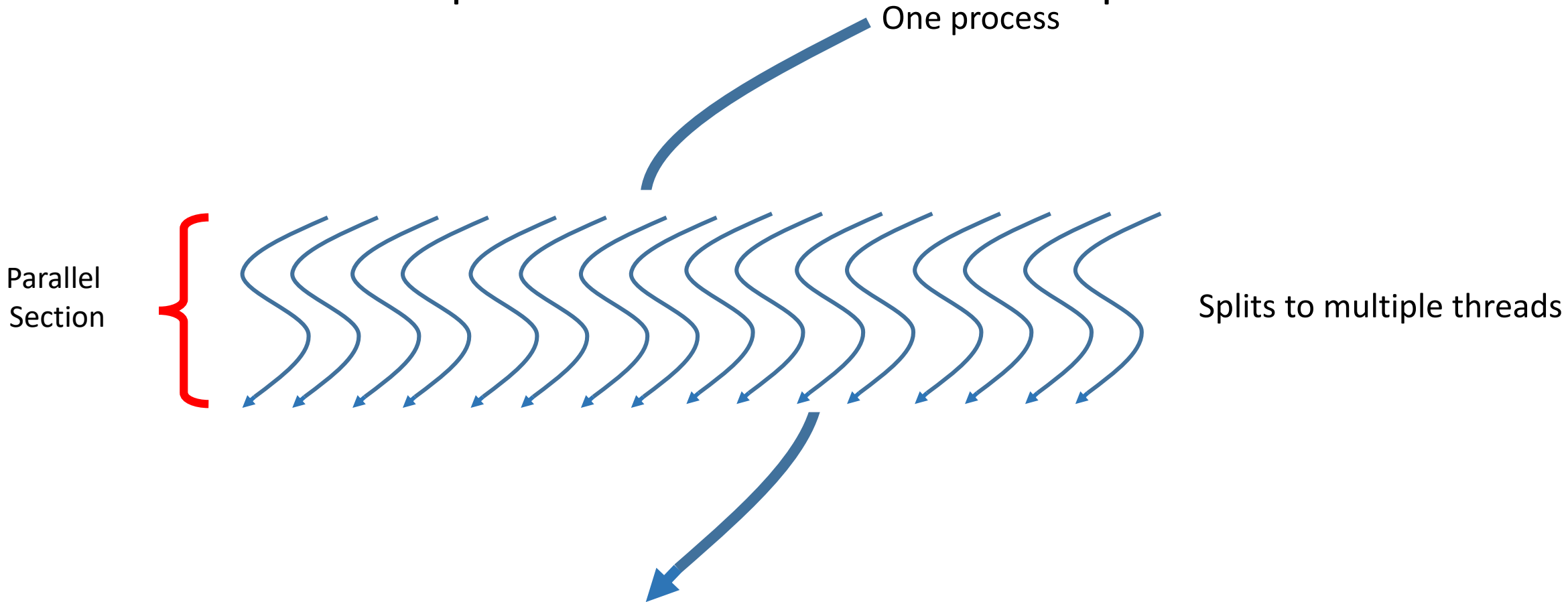

Our Program
Initial Process
Time
System call fork();
New process is created
Parallel Section
Waitpid()

Using fork()


Father
Variables:
p =987
mypid = ?
Code:
pid_t p, mypid;
…
p = fork(); p = 987
if (p < 0) {
perror("fork");
exit(1);
} else if (p == 0) {
mypid = getpid();
child();
} else {
father();
}
PID=981
Child
Variables:
p = 0
mypid = 987
Code:
pid_t p, mypid;
…
p = fork(); p = 0
if (p < 0) {
perror("fork");
exit(1);
} else if (p == 0) {
mypid = getpid();
child();
} else {
father();
}
PID=987

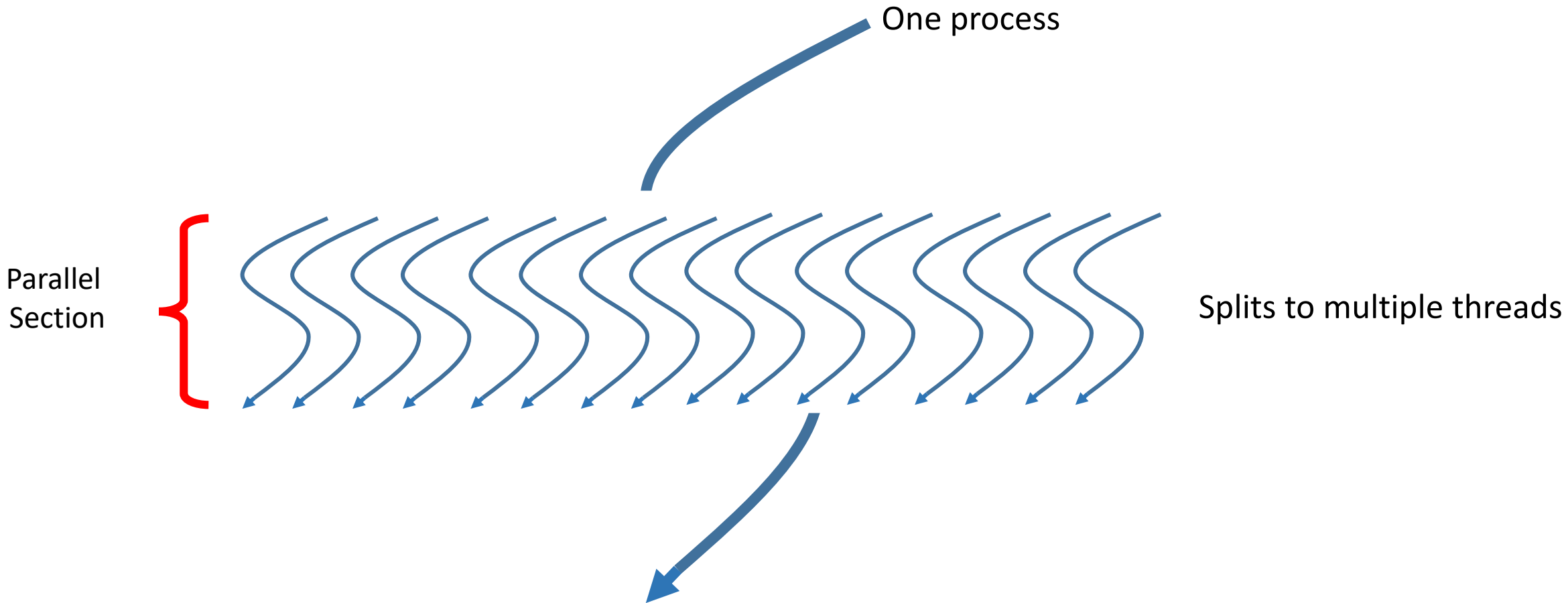
Using Threads: Pthread library

With fork() we copy the main process. Pthreads allow us to create threads that run a specific function with different inputs



Using Threads: Pthread library

All threads run the same function – different inputs



Example of using pthreads.

Assume that we have an array with 128 elements (integers) and we want to increase each element by 1. We will run this program in an 8 core machine.

Example of using pthreads.

Assume that we have an array with 128 elements (integers) and we want to increase each element by 1. We will run this program in an 8 core machine.

I need 8 threads. Each one of them will perform 16 additions

Thread 0: Increase elements from 0 – 15

Thread 1: Increase elements from 16 – 31

…

Thread 8 : Increase elements from 111 – 127

How does that function look like?

(This code is not 100% correct for many pthread specific reasons)

```
void* every_thread_function(int threadId, int * array){
  for(int i = theadId*16; i<threadId*16+16; i++){
    array[i]++;
  }
  pthread_exit(NULL);
}
```

Multithreading Programming Issues

What is the most difficult part of group projects?

Communication

Same problem for multithreading. The bank account problem

Balance = 200

Thread 0:

Balance = Balance – 100

Thread 1:

Balance = Balance +10

If thread 0 and thread 1 execute in parallel what is the value of balance at end?

Multithreading Programming Issues

Balance = 200

Thread 0:

Balance = 100

Thread 1:

Balance = 210

The actual intended result was 110…

We call this problem a race condition

Locks: A solution to race conditions

In order to access a “shared” variable you need to first acquire the lock

Example:

```
pthread_mutex_t lock; // defying a lock
pthread_mutex_lock (&lock);
//critical section
array(i)++;
//end of the critical section
pthread_mutex_unlock(&lock);
```

Locks: A solution to race conditions

Different types of locks (spinlock, mutex)

Hardware solutions (Test and Set)

Locking can be complex and even lead to dead locks:

	Thread 0:	Thread 1:
1)	pthread_mutex_t lock1	
2)		pthread_mutex_t lock2
3)	pthread_mutex_t lock2	
4)		pthread_mutex_t lock1

Question 2 Concurrency [9 pts]

Suppose you wanted to write a class that wraps a `std::stack<T>` such that it is thread safe—that is, so that multiple threads can call `push` and `pop` without any problems. If a thread calls `pop` and the stack is empty, it should block until something is pushed.

```
class ThreadSafeStack {
private:
  pthread_mutex_t lock;
  pthread_cond_t cv;
  std::stack<T>  stack;
public:
  ThreadSafeStack() {  /* assume this is written,
                          and initializes lock and cv */ }
  void push(const T& toPush) {









 }
```

Question 2 Concurrency [9 pts]

Suppose you wanted to write a class that wraps a `std::stack<T>` such that it is thread safe—that is, so that multiple threads can call `push` and `pop` without any problems. If a thread calls `pop` and the stack is empty, it should block until something is pushed.

```
class ThreadSafeStack {
private:
  pthread_mutex_t lock;
  pthread_cond_t cv;
  std::stack<T>  stack;
public:
  ThreadSafeStack() {  /* assume this is written,
                          and initializes lock and cv */ }
  void push(const T& toPush) {

  pthread_mutex_lock(&lock);        //1 point: locks mutex first
  stack.push(toPush);               //1 point: pushes on to stack
  pthread_cond_signal(&cv);         //1 point: signals cv
  pthread_mutex_unlock(&lock);      //1 point: unlocks mutex last




 }
```

```
T pop() {
```

```
T pop() {

 pthread_mutex_lock(&lock);          //1 point: locks mutex first
 while(stack.empty(){                //1 point: while loop for empty
   pthread_cond_wait(&cv, &lock);    //1 point: waits on cv
 }                                   //   (deduct 0.5 if dont pass in &lock)
 T ans = stack.top();                //1 point: get/pop
 stack.pop();                        // (fine if they do ans = stack.pop())
 pthread_mutex_unlock(&lock);        //1 point: unlock last
 return ans;




}
```

Question 6 Coding 1 [12 pts]

Write the `reverse` method in the `LinkedList` class below. This method should reverse the order of the linked list:

```
template<typename T>
class LinkedList {
  private:
    class Node{
       public:
        Node * next;
        T data;
        Node(T _data): next(NULL), data(_data) {}
        Node(T _data, Node * _next): next(_next), data(_data) {}
    };
  Node * head;
public:
  void reverse() {
```

```
void reverse() {
  Node * ans = NULL;
  while (head != NULL) {
    Node * next = head->next;
    head->next = ans;
    ans = head;
    head = next;
  }
  head = ans;
}
```

Question 8 Coding 3 [22 pts]

Suppose you have the following `BinaryTree` class:

```
template<typename T>
class BinaryTree {
  private:
    class Node {
    public:
      T data;
      Node * left;
      Node * right;
    };
    Node * root;
  const T& minInTree() {
    //already implemented, not shown
  }
  const T& maxInTree() {
    //already implemented, not shown
  }
 public:
  //constructors, destructors, other methods not shown
  bool isBSTOrdered() {
    //you will write this
  }
};
```

You must write the `isBSTOrdered` method which determines if the `BinaryTree` obeys the BST ordering. You may find the `minInTree` and `maxInTree` methods useful to do this, which determine the smallest and largest value in the tree respectively. These methods may only be called if the tree is non-empty (else they will throw an exception). You may write any helper methods you wish to (and are encouraged to write at least one).

Remember:

1)What rules does a BST have to obey?

2)Where are the initial max and min values of the BST?

Question 8 Coding 3 [22 pts]

Suppose you have the following `BinaryTree` class:

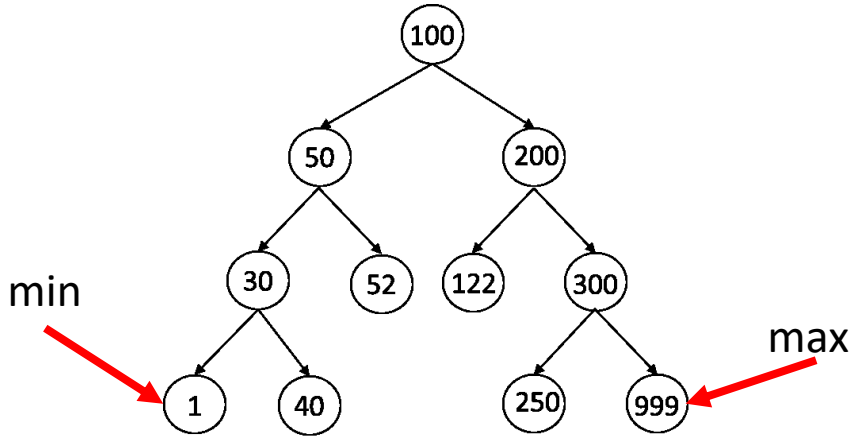
```
template<typename T>
class BinaryTree {
  private:
    class Node {
    public:
      T data;
      Node * left;
      Node * right;
    };
    Node * root;
  const T& minInTree() {
    //already implemented, not shown
  }
  const T& maxInTree() {
    //already implemented, not shown
  }
 public:
  //constructors, destructors, other methods not shown
  bool isBSTOrdered() {
    //you will write this
  }
};
```

You must write the `isBSTOrdered` method which determines if the `BinaryTree` obeys the BST ordering. You may find the `minInTree` and `maxInTree` methods useful to do this, which determine the smallest and largest value in the tree respectively. These methods may only be called if the tree is non-empty (else they will throw an exception). You may write any helper methods you wish to (and are encouraged to write at least one).

Remember:

1)What rules does a BST have to obey?

2)Where are the initial max and min values of the BST?



```
bool isBSTOrdered(Node * curr, const T& min, const T& max) {
  if (curr == NULL) {
    return true;
  }
  if (curr->data < min || curr->data > max) {
    return false;
  }
  return isBSTOrdered(curr->left,  min,        curr->data) &&
         isBSTOrdered(curr->right, curr->data, max);
}
```

```
bool isBSTOrdered() {
if (root == NULL) {
  return true;
}
return isBSTOrdered(root, minInTree(), maxInTree());
}
```