Backtracking

- It is a refinement of exhaustive search method.
- The solution is assumed to be a vector $\langle v_1, v_2, \ldots, v_m \rangle$.
- Backtracking traverses the domain of vectors representing the solution in depth-first manner.
- The algorithm would try to extend a partial solution represented by $\langle v_1, v_2, \ldots, v_i \rangle$ by adding a new element.
- If the partial solution cannot lead to solution then traversal backtracks by removing the last element and attempts to extend the solution by including another possible element.

Backtracking

```
backtrack(v, 1, i) {
    if (solution(⟨v_1, ..., v_i⟩))
        return (⟨v_1, ..., v_i⟩)
    for (each v)
        if (feasible(⟨v_1, ..., v_i, v⟩)) {
            i = i + 1;
            v_i = v;
        sol = backtrack(v, 1, i⟩);
        if (sol != ()) return sol;
    }
    return ();
}
```


Feasibility of Partial Sol

- If S_i is domain of v_i then the solution space is $S_1 \times \ldots \times S_m$.
- Validity criteria of used in checking feasibility of a partial solution determines which portion of the solution space that has to be traversed to get to the solution.
- Traversal of solution space can be represented in form of a depth-first tree of implicit graph that includes all paths.
- The graph is not stored entirely, only a path from root to current element is stored, so that back-track does not need huge storage.

Turnpike Reconstruction

- We have n points located on X-axis. Let x_i be the coordinate of point p_i.
- Let $x_1 = 0$ and points are given in left to right order.
- Let $d_1, d_2, \ldots, d_m$ represent distances between each pair of points.
- Note that there as many distances as there are distinct pair of points. So $n(n-1)/2$ distances are there. But these may not be unique.
- If points are specified then we can compute the distances in $O(n^2)$.
- But if we want to construct the points from set of given distances between point pairs then problem is not straight forward.

Turnpike Reconstruction

- Let $D = \{1, 2, 2, 2, 3, 3, 3, 4, 5, 5, 5, 6, 7, 8, 10\}$
- Since $|D| = 15$, we have $n = 6$.
- We start with placing x_1 at 0 and x_6 at 10.
- We remove 10 from D. So,

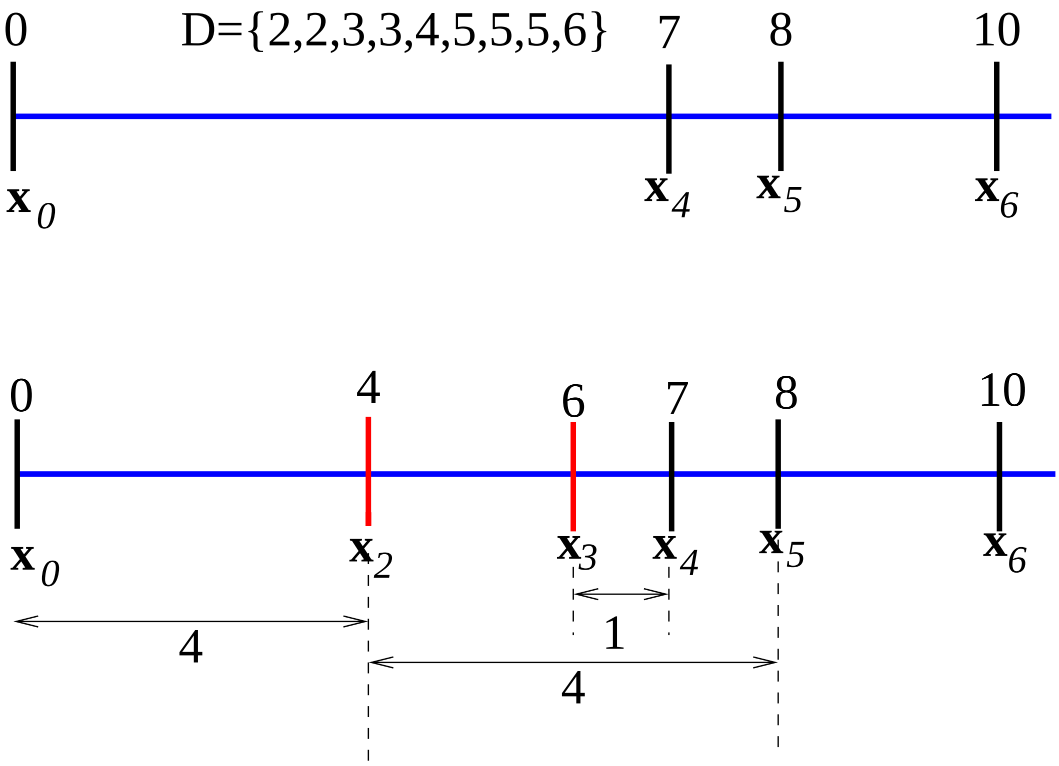
$$D = \{1, 2, 2, 2, 3, 3, 3, 4, 5, 5, 5, 6, 7, 8\}$$

- Now the largest remaining distance is 8. So either $x_2 = 2$ or $x_5 = 8$.
- Suppose we place x_5 at 8. Then remaining

$$D = \{1, 2, 2, 3, 3, 3, 4, 5, 5, 5, 6, 7\}$$

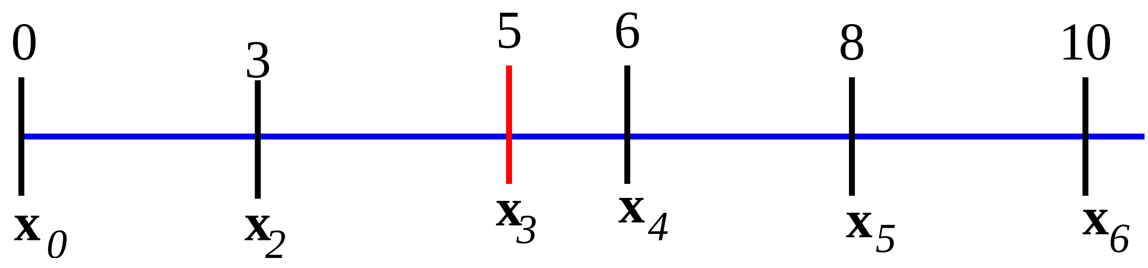
- Suppose x_4 is placed at 7 then it means $|x_6 - 7| = 3$ and $|x_5 - 7| = 1$ must be present in D.
- We find they are indeed there.
- So we may just try our luck to see if placing x_4 at 7 lead to a solution or not.

Turnpike Reconstruction



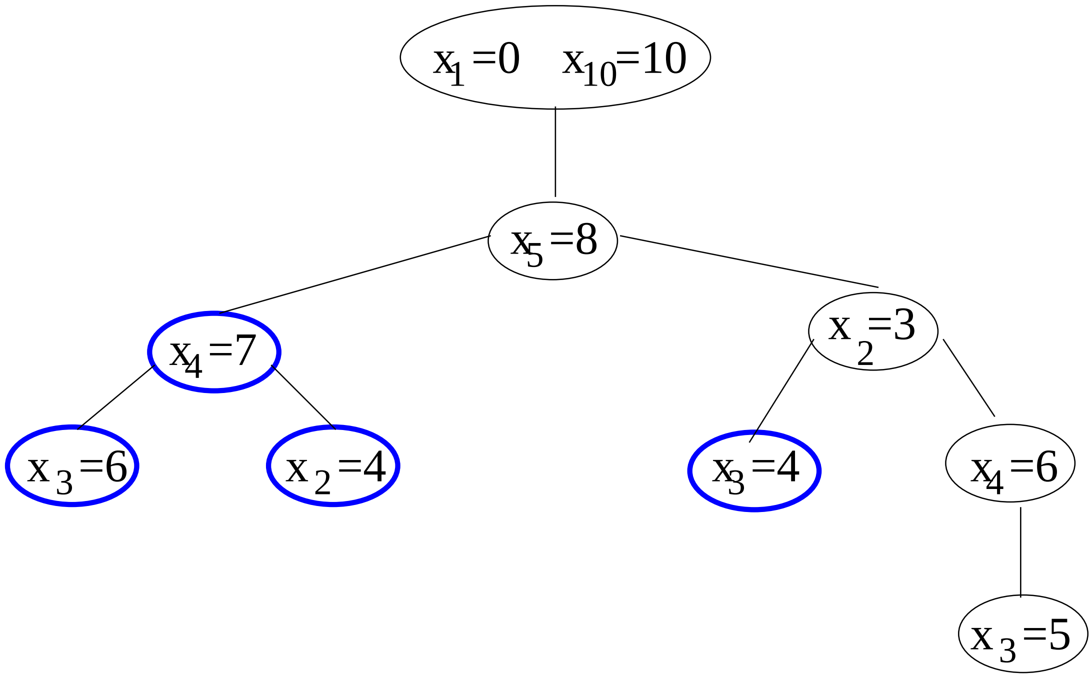
Turnpike Reconstruction

- So it is not possible to place either x_3 or x_2.
- Therefore, the previous choice of placing x_4 at 7 is wrong.
- Backtracking to that step we place x_2 at 3.
- Then the solution is found as follows.



Turnpike Reconstruction

- The decision tree representation of the search undertaken is shown below



- The nodes encircled in blue indicate inconsistent states.
- These states undone by backtrack. So we explore other nodes from x_5 and so on.

Turnpike Reconstruction

- $left$ and $right$ are boundaries of x coordinates for placing the new point. That is new x to be placed at x_{left} or x_{right}.
- D empty (or $left > right$) the solution has been found.
- First attempt will be to place $x_{right} = D_{max}$.
- If all distances generated as a result of placing the point at x_{right} are present and correct then we can tentatively place that point
- Remove all appropriate distances.
- Now try to fill from $left$ to $right - 1$.
- Otherwise, we try placing $x_{left} = x_n - D_{max}$ and repeat the same two steps.
- If none works then backtrack by removing appropriate x's and inserting all appropriate distances back to D array.

Turnpike Reconstruction

```
place(x, D, N, left, right) {
   if (isempty(D))
      return (true);
   D_max=findMax(D);
   if (|x[j] − D_max| ∈ D
         ∀1 ≤ j < left, right < j ≤ N) {
      x[right] = D_max;
      for (1 ≤ j < left, right < j ≤ N)
         remove(|x[j] − D_max|, D);
      found = place(x, D, N, left, right − 1);
      if (!found)
         for (1 ≤ j < left, right < j ≤ N)
            insert(|x[j] − D_max|, D);
   }
```

Turnpike Reconstruction

```
    if (!found && (|x[N] − D_max − x[j]| ∈ D
            ∀1 ≤ j < left, right < j ≤ N) ) {
        x[left] = x[N] − D_max;
        for (1 ≤ j < left, right < j ≤ N)
            remove(|x[j] − D_max − x[j]|, D);
        found = place(x, D, N, left + 1, right);
        if (!found) {
            for (1 ≤ j < left, right < j ≤ N)
                insert(|x[j] − D_max − x[j]|, D);
        }
    }
    return found;
}
```


Turnpike Reconstruction

```
turnpike(x, D, N) {
   x[1] = 0;
   x[N] = deletMax(D);
   x[N-1] = deletMax(D);
   if (x[N]-x[N-1] ∈ D) {
      remove(x[N]-x[N-1], D);
      return place(x, D, N, 2, N-2);
   }
   else
   return false;
}
```

Backtracking

- If we never backtrack then there are at most $O(n^2)$ operation involving D, these are deletions and finds.
- Each call to `place` uses at most $2n$ finds.
- We can keep D as a balanced BST then deletions can be done in $\log n$ time.
- So overall time is O($n^2 \log n$).
- Since there is no limit to backtracks that can happen we can not give an analysis of the running time in presence of backtrack.
- But usually backtracks are O(1) in numbers. So it is possible that the running time is no worse than O($n^2 \log n$).

