Dynamic Programming

Algorithms

Topics

- Dynamic Programming
  - Fibonacci solution via Divide and Conquer
  - Introduction to Dynamic Programming
  - Shortest Path
  - Longest Common Subsequence
  - Knapsack

Divide and Conquer

- Divide the problem into smaller subproblems
- Conquer the subproblems recursively until the subproblem is small enough to solve
- Example Binary Search, Merge Sort

Binary Search

- Binary Search (array A, first, last, v)
  - If (last <first) output -1
  - Mid = (last + first) /2
  - If A[mid]= v   output mid
  - If A[mid]>v
    - Binary_search (A, first, mid-1,v)
  - Else Binary_search (A, mid+1, last,v)

Fibonacci Example

Fibonacci Numbers

- F(n)
  - If (n>=2) F(n)=F(n-1)+F(n-2)
  - If (n=1) F(n)=1
  - If (n=0) F(n)=0
- Running Time
- T(n)=T(n-1)+T(n-2)
  - =T(n-2)+T(n-3)+T(n-2)>=2T(n-2)
  - $T(n) >= 2T(n-2) >= 2^2 T(n-4) >= 2^k T(n-2k)$
  - So when k= n/2. $T(n) >= 2^{n/2} T(1)$, exponential

Introduction to Dynamic Programming

- “Programming” refers to a organized set of activities
  - Example Graduation program, TV program
- Stores the results for small subproblems, and looks them up instead of recomputing
- Used for recursive algorithms which would solve many subproblems repeatedly
- Usually applied to ***optimization problems***
  - Problems where there are many solution, but one is looking for the best (either maximum or minimum)
  - Ex: Shortest path between node a and node b

Shortest Path

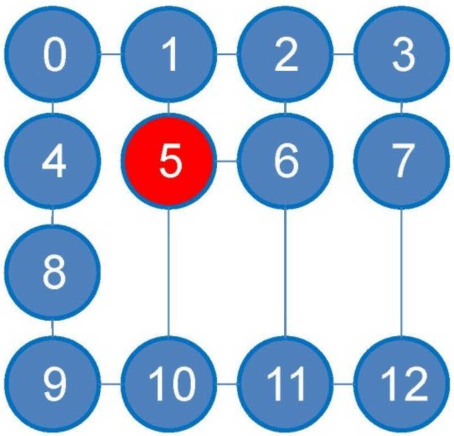
- Given a G=(V,E) and W edge weights of the edges and vertices s, t. Find the shortest path from s to t.
- d(s,t)=distance of shortest path from s to t

Shortest Path

- Un-weighted graph: BFS (breadth first search) (as opposed to Depth-First Search)
- Single Source Shortest Path (SSSP)
- O(|E|+|V|)

BFS

Example (1)

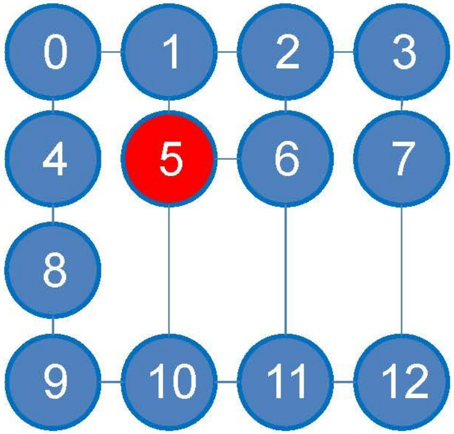


Q = {5}

D[5] = 0

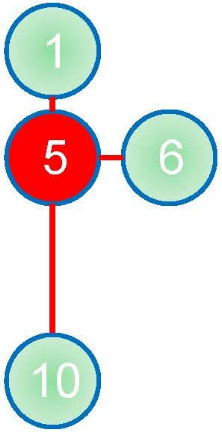
BFS

Example (2)



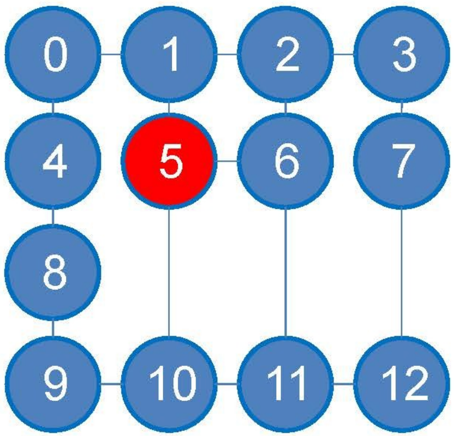
Q = {5}
Q = {1, 6, 10}

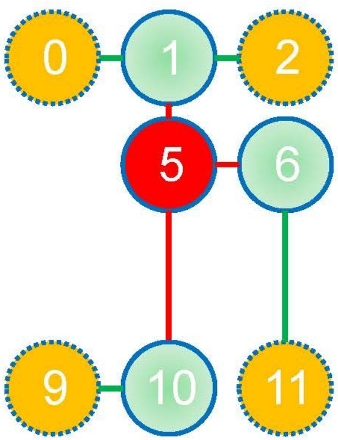
D[5] = 0
D[1] = D[5] + 1 = 1
D[6] = D[5] + 1 = 1
D[10] = D[5] + 1 = 1



BFS

Example (3)



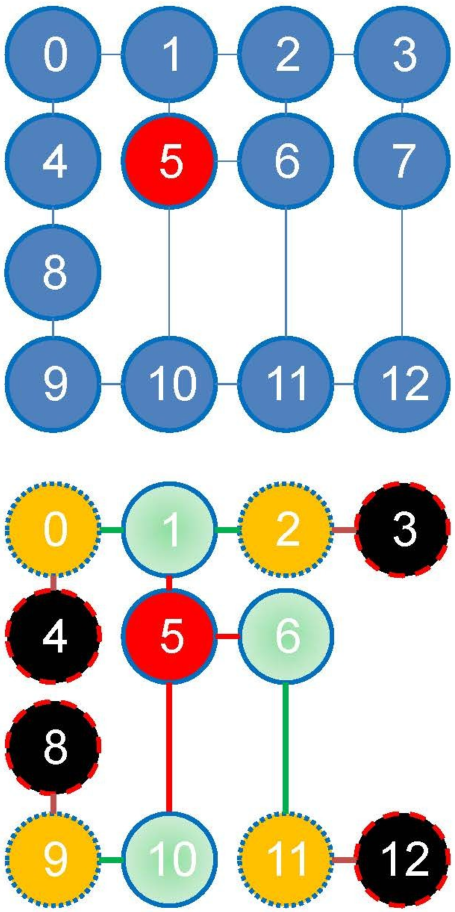


Q = {5}
Q = {1, 6, 10}
Q = {6, 10, **0, 2**}
Q = {10, 0, 2, **11**}
Q = {0, 2, 11, **9**}

D[5] = 0
D[1] = D[5] + 1 = 1
D[6] = D[5] + 1 = 1
D[10] = D[5] + 1 = 1
D[0] = D[1] + 1 = 2
D[2] = D[1] + 1 = 2
D[11] = D[6] + 1 = 2
D[9] = D[10] + 1 = 2

BFS

Example (4)

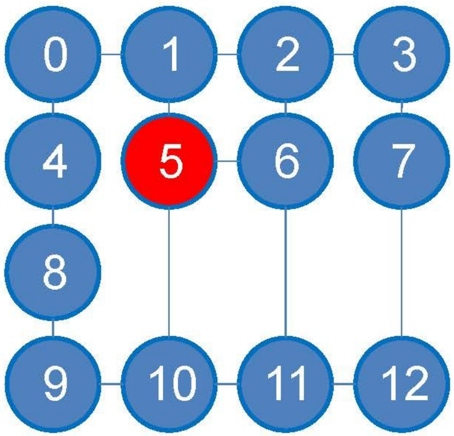


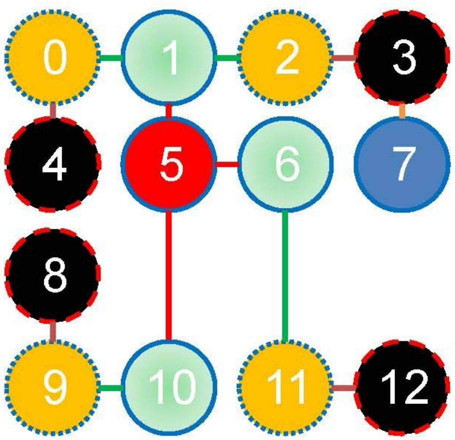
Q = {5}
Q = {1, 6, 10}
Q = {6, 10, **0, 2**}
Q = {10, 0, 2, **11**}
Q = {0, 2, 11, **9**}
Q = {2, 11, 9, **4**}
Q = {11, 9, 4, **3**}
Q = {9, 4, 3, **12**}
Q = {4, 3, 12, **8**}

D[5] = 0
D[1] = D[5] + 1 = 1
D[6] = D[5] + 1 = 1
D[10] = D[5] + 1 = 1
D[0] = D[1] + 1 = 2
D[2] = D[1] + 1 = 2
D[11] = D[6] + 1 = 2
D[9] = D[10] + 1 = 2
D[4] = D[0] + 1 = 3
D[3] = D[2] + 1 = 3
D[12] = D[11] + 1 = 3
D[8] = D[9] + 1 = 3

BFS

Example (5)





Q = {5}
Q = {1, 6, 10}
Q = {6, 10, **0, 2**}
Q = {10, 0, 2, **11**}
Q = {0, 2, 11, **9**}
Q = {2, 11, 9, **4**}
Q = {11, 9, 4, **3**}
Q = {9, 4, 3, **12**}
Q = {4, 3, 12, **8**}
Q = {3, 12, 8}
Q = {12, 8, **7**}
Q = {8, 7}
Q = {7}
Q = {}

D[5] = 0
D[1] = D[5] + 1 = 1
D[6] = D[5] + 1 = 1
D[10] = D[5] + 1 = 1
D[0] = D[1] + 1 = 2
D[2] = D[1] + 1 = 2
D[11] = D[6] + 1 = 2
D[9] = D[10] + 1 = 2
D[4] = D[0] + 1 = 3
D[3] = D[2] + 1 = 3
D[12] = D[11] + 1 = 3
D[8] = D[9] + 1 = 3
D[7] = D[3] + 1 = 4

This is the **BFS = SSSP ☺ spanning tree** when BFS is started from vertex 5

Floyd-Warshall Shortest Path

- For edge e=(i,j)

  Set d(e)=infinity if e is not an edge

  otherwise set d(e)=w(e)
- For k=1 to n
  - For i = 1 to n
    - For j= 1 to n
      - $d(i,j)= \min \{d(i,j),d(i,k)+d(k,j)\}$
- n^3 running time (n<100): Floyd-Warshall
- All pairs shortest paths: APSP
- Can have negative weight

Bellman Ford

- Single Source Shortest Path: SSSP
- $O(|V|^2*|E|)$ if $E=O(|V|^2)$ what is running time
- May have negative weight cycle

Dijkstra’s Shortest Path

- Most famous (Dijkstra’s algorihtm)
- Finds shortest path from a given source s to all other vertices: SSSP
- Very similar to Prim’s Algorithm
- Greedy Algorithm
- Fibonacci heap: $O(|E|+ |V|\lg |V|)$
- Min priority queue $O(|V|^2)$
- OSPF (open shortest path first) protocol in networking
- Special implementation for sparse graphs
- How do you store a graph?

Shortest Path

- Input, G=(V,E,W) source s
- Output d[v] = best distance from s to v
- Set $d[v]=\infty$ for all $v \neq s$
- Set d[s]=0, S= { }   Q=V
- While Q is not empty
  - Find u which satisfies min $\{d[v], v \in Q\}$
  - S= S U {u}, Q=Q-{u}
  - For each v adjacent to u, i.e. $(u,v) \in E$
    - d[v]= min{ d[v], d[u]+W(u,v)}

Longest Common Subsequence

- Given two string, find the longest common subsequence
- Example
  - S=accg**gtc**ga**gt**g**c**gc**ggaagccggccgaa**
  - T=**gtcgt**t**cggaa**t**gccg**tt**gc**t**c**t**gt**t**aa**a
  - Longest subsequence gtcgtcggaagccggccgaa
- Similar to Diff command in Unix
- Used for file revisions or remote updates, spelling corrections, plagiarism detection, speech recognition, comparing two DNA sequences

Longest Common Subsequence

- Theorem

  Let $X=<x_1, x_2,\ldots,x_m>$ $Y=<y_1,y_2,\ldots y_n>$, and $Z=<z_1,z_2,\ldots z_k>$ be a LCS of X and Y

  – If $x_m=y_n$, then $z_k=x_m=y_n$ and Z_{k-1} is a LCS of X_{m-1} and Y_{n-1}

  – If $x_m \neq y_n$ and $z_k \neq x_m$, then Z is a LCS of X_{m-1} and Y_n

  – If $x_m \neq y_n$, and $z_k \neq y_n$ then Z is a LCS of X_m and Y_{n-1}

LCS

- Define c[i.j] to be the length of the LCS of X_i and Y_j
- What is c[0,j]?
  - 0
- What is c[i,0]?
  - 0
- What is c[i,j], if i,j >0 and $x_i=y_j$?
  - c[i-1,j-1]+1
- What is c[i,j], if i,j >0 and $x_i \neq y_j$?
  - Max{ c[i,j-1],c[i-1,j] }

LCS

- LCS-Length(X,Y)
  - m= length(X), n= length(Y)
  - Set c[i,0]=0 for all I from 1 to m
  - Set c[0,j]=0 for all j from 1 to n
  - For i is 1 to m
    - For j=1 to n
      - If xi=yj
        - Then c[i,j]=c[i-1,j-1]+1, and b[i,j]="↖"
        - else if c[i-1,j]>= c[i,j-1]
        - then c[i,j]=c[i-1,j] and b[i,j]="↑"
        - else c[i,j]=c[i,j-1] and b[i,j]="←"

yj xi	0	1 B	2 D	3 C	4 A	5 B	6 A
0	0	0	0	0	0	0	0
1 A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2 B	0						
3 C	0						
4 B	0						
5 D	0						
6 A	0						
7 B	0						