Week 8: Lecture 15 - Lecture 16

Lecture 15

#graph

Dijkstra's algo

// code implementation with priority queue

// code implementation without priority queue

Problem 1

Problem - 20C - Codeforces

Problem 2

CSES - Flight Discount

idea: use Dijkstra's algo to find the min cost route. Then half the most expensive flight on the route.

Actually this idea doesn't work.

Counter example: there are two routes from source to destination

Route 1:

Src ➜ A ➜ B ➜ dst

Cost (src, A) = 4

Cost (A, B) = 2

Route 2:

Src ➜ dst

Cost (src, dst) = 5

By the idea we would have chosen route 1 and half the route from src to A, which gives final result = 2 + 2 = 4

However if we just half cost (src, dst), we would have gotten 2 instead.

Soln:

- Run Dijkstra from src ➜ dst
- Create reverse graph G'. Run Dijkstra from dst ➜ src
- for each edge e (u, v), half the cost (u, v). Find out cost from src ➜ u in G and cost from v ➜ dst in G'
- Take the min over all possible results

Lecture 16

#dp

Key to solve dp problems:

- Finding parameters that uniquely define a problem *"DP State"*

  E.g. in merge sort parameters are: (start index, end index)
- Dividing problems into overlapping subproblems

Problem 1

CSES - Dice Combinations

idea: ways(1) = 1; ways(n) = ways(n-1) + ways (n-2) + …. + ways(n-6)

Example:

ways(3) = 1 (last selection) + ways (2); – actually 2 ways

2 (last selection) + ways (1);

3 (last selection) + ways(0);

Results = combine all three situations above = 4

DP initialization

dp[0] = 1

dp[1] = 1

Core code:

```
vector <int> dp(n+1);
dp[0] = 1;
dp[1] = 1;
int ans = 0;
for (int i = 2; i <= n; ++i) {
  for (int j = 1; j <= 6; j++) {
      if (i - j >= 0) {
          ans += dp[i-j];
      }
  }
}
```

Problem 2

CSES - Minimizing Coins

Recurrence relation:

coins(n) = min {coins(n-i)} for all i + 1

Problem 3

CSES - Book Shop

For each book there are two decisions to make: to purchase or not to purchase

f(n, i) represents max # of pages possible with budget = n considering book i

f(n, i) = max {f(n-price[I], i-1) + pages[i], f(n, i-1)} // the second situation represents when current book is skipped.

// code to do