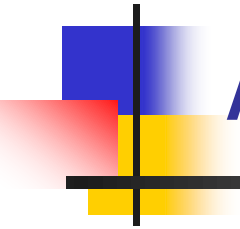
CMSC 501



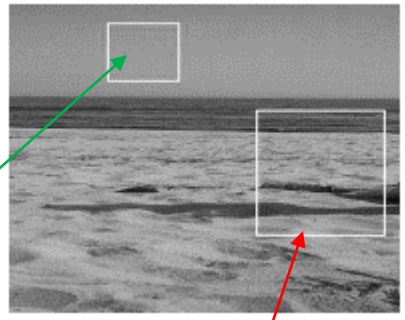
Advanced Algorithms

Homework Project

Date due:
Wednesday, November 16, noon

Problem: Image segmentation

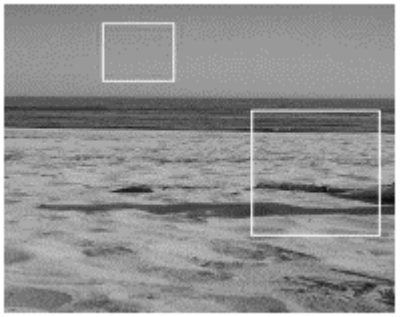
- You have a gray-scale image
  - An object
  - And background



- Someone pointed to several pixels and said:
  - These pixels very likely belong to the object
- And pointed to several other pixels and said:
  - These pixels very likely belong to the background

Problem: Image segmentation

- For each pixel *i* we define in some way:
  - f_i – penalty if the pixel ends up in foreground
  - b_i – penalty if the pixel ends up in background
- For each pair of pixels *i*, *j* we define in some way:
  - p_{ij} – penalty if pixels *i*, *j* are classified differently, that is, *i* ends up in the foreground and *j* ends up in the background
- All penalties are real numbers >=0
- Example:
  - For pixels marked by user as foreground, we can set f_i to 0, b_i to high value
  - For pixels marked by user as background, we can set f_i to high value, b_i to 0
  - For neighboring pixels of similar gray level, we can set p_{ij} to a high value (we really want them together, either both in foreground, or both in background)



Problem: Image segmentation

- For each pixel *i* we define in some way:
  - f_i – penalty if the pixel ends up in foreground
  - b_i – penalty if the pixel ends up in background
- For each pair of pixels *i*, *j* we define in some way:
  - p_{ij} – penalty if pixels *i*, *j* are classified differently, that is, *i* ends up in the foreground and *j* ends up in the background
- The task is to classify each pixel as background ($i \in B$) or foreground ($i \in F$)
  - that is, partition the set of pixels into sets F, B
- in a way that minimizes the total penalty:
  - $L(B,F) = \sum_{i \in B} b_i + \sum_{i \in F} f_i + \sum_{i \in F,\, j \in B} p_{ij}$
- Different breakdown of the image into B and F results in different L(B,F), we aim to pick B/F that minimize L(B,F)

Problem: Image segmentation

- For each pixel *i* we define in some way:
  - f_i – penalty if the pixel ends up in foreground
  - b_i – penalty if the pixel ends up in background
- For each pair of pixels *i*, *j* we define in some way:
  - p_{ij} – penalty if pixels *i*, *j* are classified differently, that is, *i* ends up in the foreground and *j* ends up in the background
- Details for setting up the penalty depend on the specific application, e.g. type of image
  - The algorithm should work (better or worse) no matter how we choose the penalties, as long as they're all >=0

Parametrization

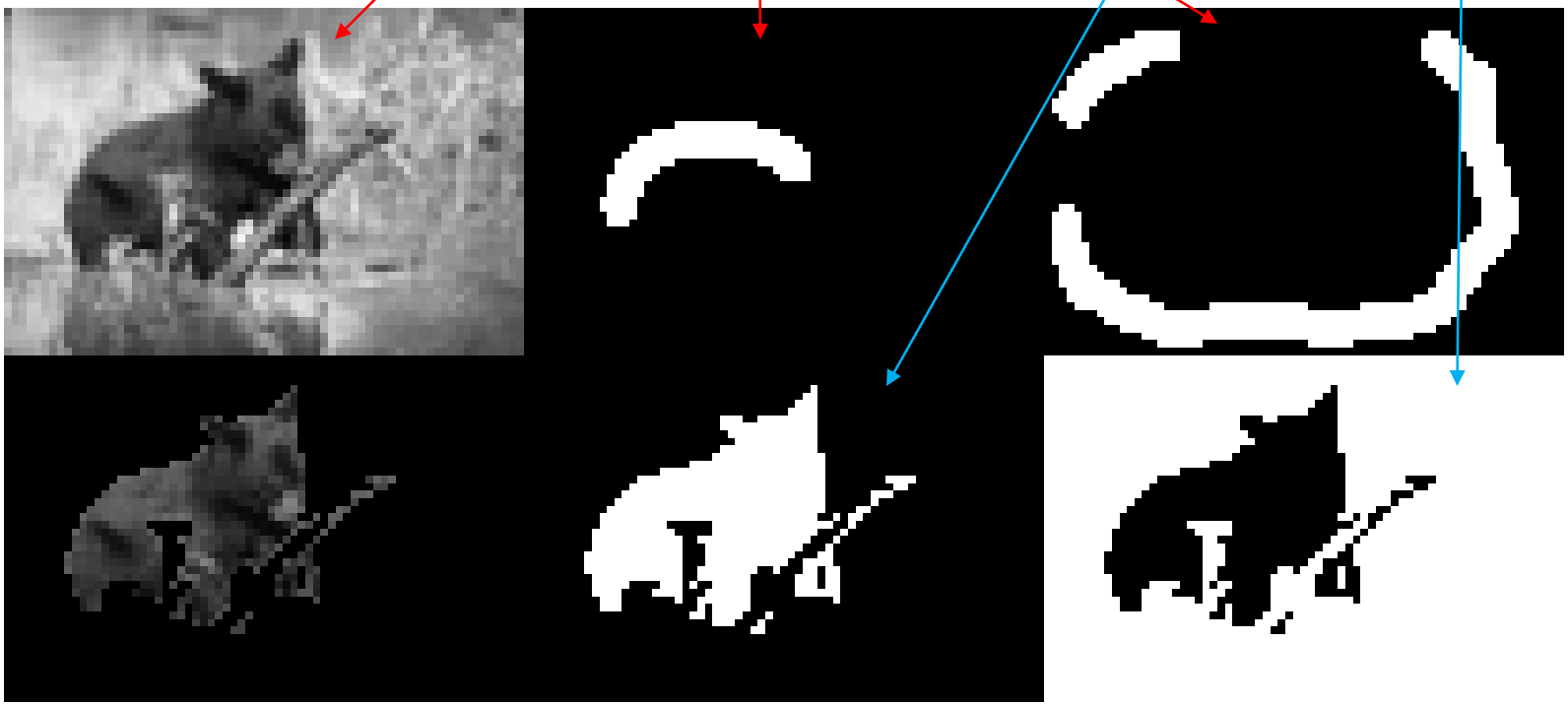
- For each pixel *i* we define in some way:
  - f_i – penalty if the pixel ends up in foreground
    - E.g. how different is the pixel color (gray-level) from mean color of all pixels provided as "known background"
  - b_i – penalty if the pixel ends up in background
    - E.g. how different is the pixel color (gray-level) from mean color of all pixels provided as "known background"
- For each pair of pixels *i*, *j* we define in some way:
  - p_{ij} – penalty if pixels *i*, *j* are classified differently, that is, *i* ends up in the foreground and *j* ends up in the background
    - e.g. 0 if pixels are more than X apart, and if they're within X distance, a penalty inversely depending on distance, and inversely depending on color similarity
- See Java file on Blackboard for how to set it up
  - Your java function for the segmentation solver should take as argument a Parametrization object that provides the penalties for given i,j
  - So that you can reuse the same solver with different way of setting penalties

Input/output

- Your Java program should accept 5 command line parameters
  - java segmentApp img.png fgIn.png bgIn.png fgOut.png bgOut.png
    - Input:
      - img.png is the image to be segmented
      - fgIn.png is the image where "known foregroung/object" pixels are in white, the rest is black
      - bgIn.png is the image where "known background" pixels are in white, the rest is black
    - Output:
      - Should be written to fgOut.png and bgOut.png
      - fgOut.png has pixels that are classified as "foreground/object" in white, and the pixels classified as "background" in black
      - bgOut.png has pixels that are classified as "foreground/object" in black, and the pixels classified as "background" in white
    - All five images should be 8-bit grayscale images, of the same size (height x width)

Input/output

- Your Java program should accept 5 command line parameters
  - java segmentApp img.png fgIn.png bgIn.png fgOut.png bgOut.png



Solution details

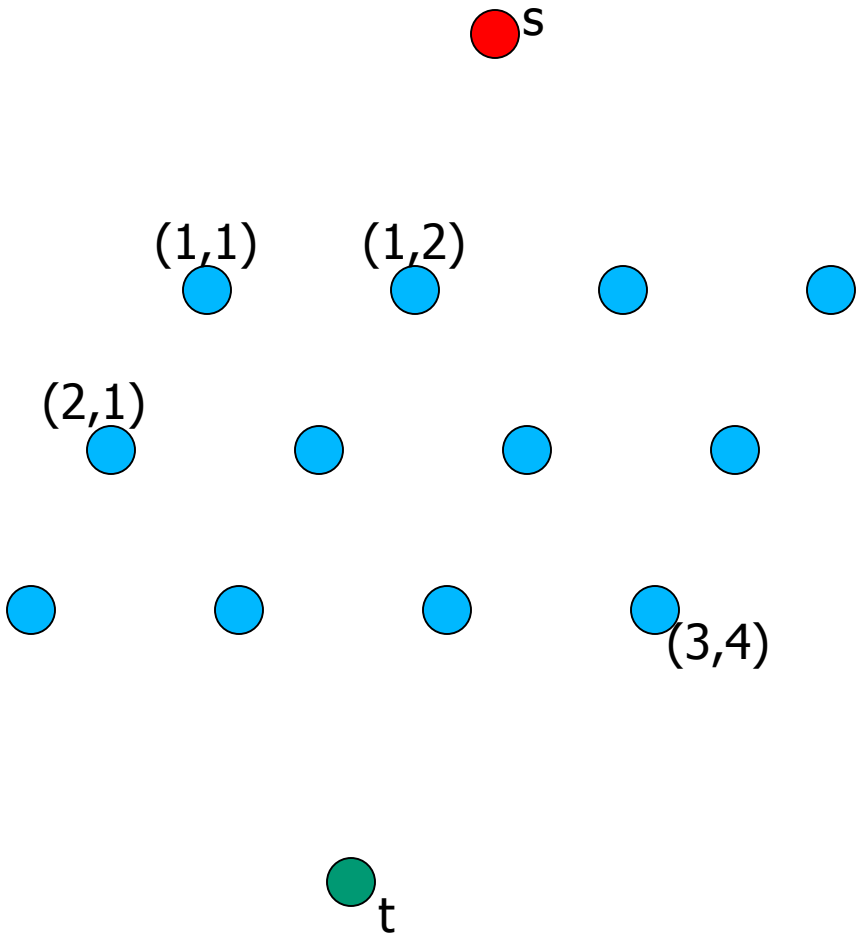
- Homework assignment should be done individually
  - In Java
  - Only Java code written by yourself (no web/friend/textbook/etc. Java code)
  - You can rely on high-level, **abstract** pseudocode (from 501 slides or from other sources, **you need to specify the sources you consulted in description.txt**), but **the actual Java code has to be designed & written by you**
- You may use:
  - built-in Java simple data structures
    - arrays, hashes, lists, queues, priority queues
- You are not allowed to use:
  - libraries of graph data structures or algorithms for graph problems

Returning the homework

- Submit via Blackboard
  - Java code
  - A short description of your approach in a Word/PDF/txt file

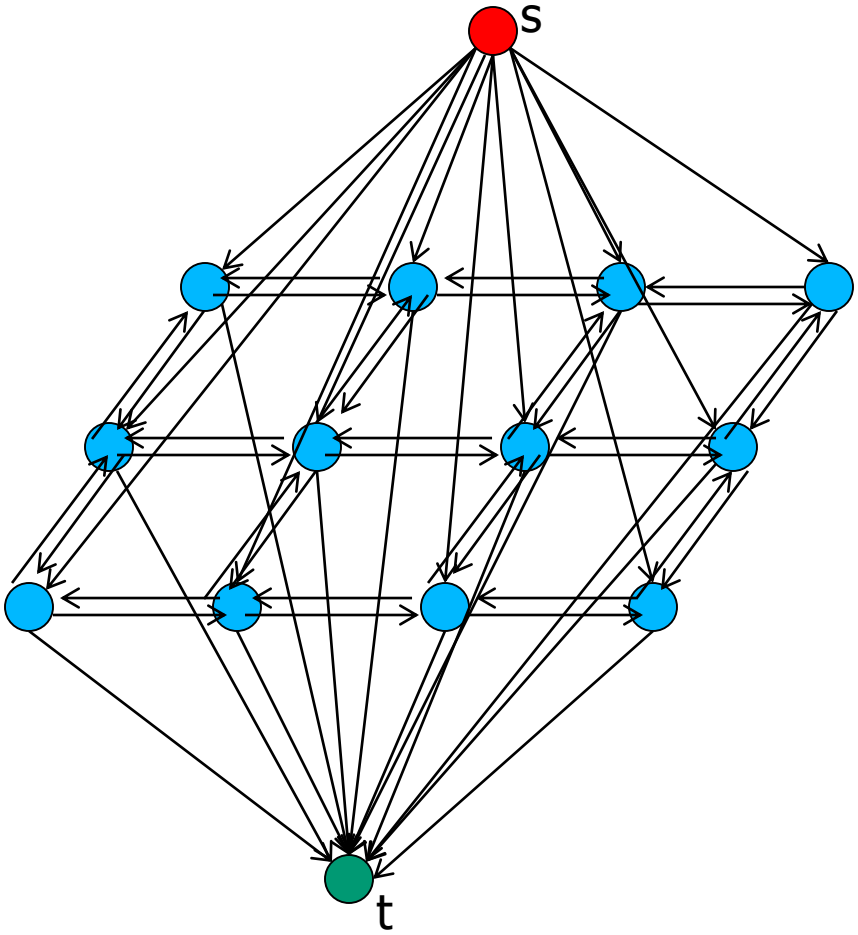
Hints

- From 3 by 4 image to flow net with 14 vertices



Hints

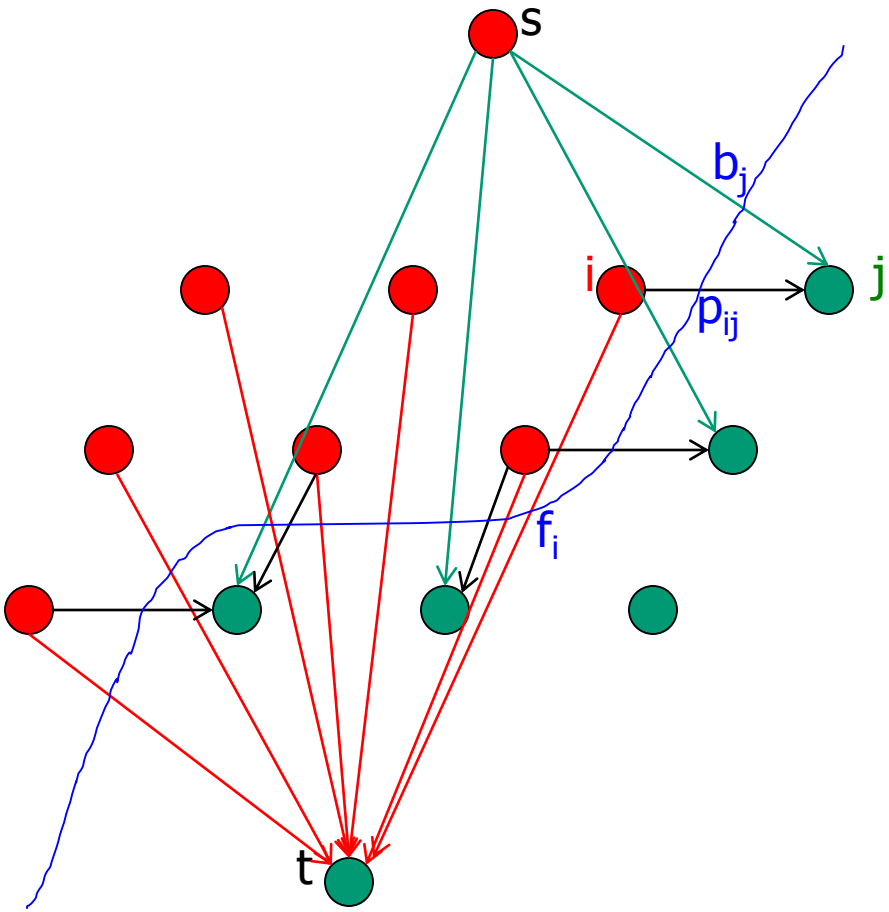
- From 3 by 4 image to flow net with 14 vertices



- All pixel-pixel pairs have edges in both directions!
  - Adjust max-flow equations/pseudocode accordingly!

Hints

- Capacity of an S|T cut ⇔ penalty for an F/B partitioning

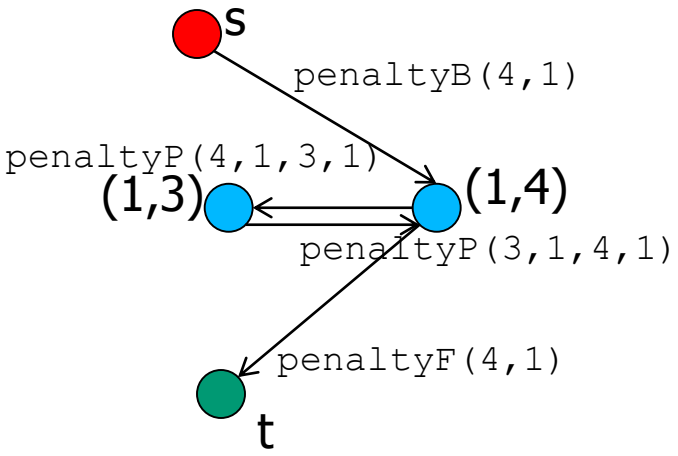


Penalty L(B,F) $= \Sigma_{i\epsilon B} b_i + \Sigma_{i\epsilon F} f_i + \Sigma_{i\epsilon F,\ j\epsilon B} p_{ij}$

(S|T) cut capacity= $\Sigma_{j\epsilon T} c(s,j) + \Sigma_{i\epsilon S} c(i,t) + \Sigma_{i\epsilon S,\ j\epsilon T} c(i,j)$

Hints

- Use these functions to obtain penalties for each edge

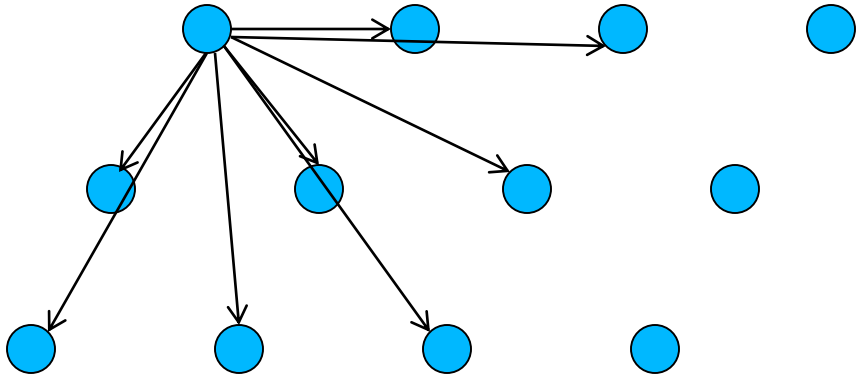



```
interface Parametrization
{
     public void initialize(int img[][], int height, int width, int bg[][], int fg[][]);
     public double penaltyP(int x1, int y1, int x2, int y2);
     public double penaltyF(int x, int y);
     public double penaltyB(int x, int y);
}
```

- There's a class SomeParametrization that implements this interface, and it includes specific formulas for each type of penalty. You need to initialize it!
  - `myParam=new SomeParametrization();`
    `myParam.initialize(img, imH, imW, fgIn, bgIn);`
  - It assumes all images are grayscale, with 0=black, 255=white
  - img[0...height-1][0...width-1], same for fg[][] and bg[][], i.e. first array index is Y, second is X

Hints

- Pixel vertex may be connected to many other pixel vertices, so you need to check all pairs
  - Any pair i->j is connected if $p_{ij}>0$



  - In practice, expect a small neighborhood (e.g. up to 3 pixels in each direction, see penaltyP in Java)
    - So graph will be sparse => use adjacency list representation, not adjacency matrix

- In case of any questions, don't hesitate to contact me